C для профессиональных программистов

Москва, 1989 г.

С О Д Е Р Ж А Н И Е

Предисловие

Если вы хотите создавать программы мирового уровня, написанные на Си, то эта книга - для вас!

Меня зовут Герб Шилдт. Я ветеран программирования во многих компаниях и программирую на Си более десяти лет. Перед тем, как начать писать эту книгу, я изучил большое количество удачного программного обеспечения, пытаясь определить, какие черты они имеют в отличие от менее удачных программ. Я хотел понять, что делает одни программы более удачными, чем другие, аналогичные им. После этого прояснились общие черты. Удачные программы были написаны людьми, которые не только имели крепкую хватку в специальной области, но и в совершенстве освоили оборудование компьютера, включая операционную систему и аппаратное обеспечение. Только программист, осуществляющий полный контроль над ними может писать программы с дружелюбным интерфейсом пользователя которые эффективно выполняются и дают пользователям большую гибкость.

Эта книга открывает многие секреты, используемые мастерами программирования для достижения профессиональных результатов. С ее помощью вы расширите подходы и методы, которые делают программы интересными. После прочтения книги вы будете способны писать программы, которые заслужат внимание. Здесь

рассматриваются следующие вопросы:
Прямой доступ к памяти экрана для быстрого отображения
Исчезающие и иерархические (popup и pulldown) меню
Процедуры работы с окнами
Завершение программ и оставление их в памяти
Интерфейс с мышью
Графические функции, включая вращение объектов
Языковые интерпретаторы
Передача файлов через последовательный порт

Эта книга для любого и каждого программиста на Си, от новичка до профессионала. Даже если вы начинающий, вы можете использовать функции и программы из этой книги без понимания отдельных деталей их работы. Более подготовленные читатели могут использовать эти программы как основу для своих приложений.

Исходные тексты этой книги соответствуют стандарту ANSI, кроме некоторых функций, специфичных для ПК. Таким образом все эти программы можно компилировать на любом компиляторе, который поддерживает стандарт. Я использовал для их разработки Турбо Си и Microsoft Си.

Некоторые из многих полезных и интересных функций и программ, содержащихся в книги достаточно длинны. Если вы, как и я, возможно хотите использовать их, но вам лень набивать их на компьютере и потом выискивать неизбежные опечатки, то для такого случая я предоставляю исходные коды, содержащиеся в этой книге,

на дискете. Цена услуги - 24.95$. Заполните приложенную форму и пошлите ее по указанному адресу вместе с оплатой. Можно использовать карточки MasterCard и Visa.

ГЛАВА 1.

Исчезающие и иерархические меню.

Одна из наиболее очевидных черт профессионально написанных программ - это использование исчезающих и иерархических меню. При правильном использовании, эти меню дают программам дружелюбие, которое пользователи от них и ожидают. Хотя по существу и простые, и исчезающие, и иерархические меню представляют некоторые трудности в программировании.

Создание исчезающих и иерархических меню требует прямого управления экраном. Хотя основные программы меню полностью мобильны, программы доступа к экрану зависят от операционной системы и оборудования и не используют обычные функции Си ввода/вывода на консоль. Программы видео доступа разработаны для работы с любым компьютером, использующим ДОС и имеющим BIOS операционной системы совместимый с IBM. BIOS-ДОС выбран потому что он широко используется, но вы можете применить основные идеи и в других системах.

Даже если вас сейчас не интересуют исчезающие и иерархические меню, то вам следует прочитать часть этой главы, в которой обсуждаются видеоадаптеры, знание многих основных идей необходимо для понимания последующих глав.

Что такое исчезающие и иерархические меню?

Важно понимать что такое исчезающие и иерархические меню и чем они отличаются от стандартных меню. При использовании стандартных меню экран очищается или сдвигается, и появляется меню. Когда выбор сделан, экран опять очищается или сдвигается и программа продолжается. Выбор выполняется по номеру или по первой букве каждой альтернативы.

Когда используется исчезающее или иерархическое меню, то оно покрывает прямо содержимое экрана. После выбора режима, экран возвращается в предыдущее состояние. Вы выбираете нужный режим из меню одним из двух способов: (1) нажимая активную клавишу, которая является буквой или номером, связанным с выбором, или (2)

используя клавиши управления курсором для передвижения подсвеченного поля и клавишу Ввод. Обычно текущее поле показывается в инверсном виде. Основная разница между стандартными меню и исчезающими и иерархическими меню в том, что стандартное меню прерывает программу. Исчезающие и иерархические меню только приостанавливают текущие действия программы. С точки зрения пользователя стандартное меню - прерывание концентрации, тогда как исчезающее меню - просто легкая приостановка, концентрация внимания пользователя не нарушена.

Разница между исчезающими и иерархическими меню проста. Только одно исчезающее меню может быть на экране в данный момент времени. Оно используется когда меню имеет только один уровень в глубину, это бывает, когда выбор из меню не имеет подвыборов. С другой стороны несколько иерархических меню могут быть активны одновременно. Они используются когда выбор из одного меню может потребовать использования другого меню для определения некоторых альтернатив. Например, вы можете использовать иерархическое меню, если вы пишете программу, которая определяет фрукт. Если пользователь выбрал "яблоко", следующее меню предлагает выбрать цвет яблока, а третье меню высвечивает яблоки, которые удоволетворяют предыдущим выборам.

Вы можете представлять исчезающее меню просто как иерархическое меню, которое не имеет подменю, но разработка отдельных процедур для этих типов меню имеет то преимущество, что иерархическое меню требует значительно более сложной программы, чем простое исчезающее меню.

Хотя имеется много способов расположения меню на экране, функции, разработанные в этой главе имеют наиболее общий вид. Этот метод помещает очередное поле меню на новую строку под первым полем.

Работа видеоадаптеров.

Из-за того, что создание исчезающих и иерархических меню требует прямого управления экраном, важно понимание адаптеров дисплея. Три основных типа адаптеров - это одноцветный адаптер, цветной/графический адаптер (CGA) и усовершенствованный графический адаптер (EGA). CGA и EGA могут иметь несколько режимов работы, включая 40- или 80- символьный текст или графические операции. Эти режимы показаны в таблице 1-1. Программы меню, разработанные в этой главе, разработаны для использования режима 80-символьного текста, который является наиболее общим режимом для общецелевых применений. Это значит, что видео режим системы должен быть 2, 3 или 7. Независимо от используемого режима - координаты левого верхнего угла - 0,0.

Таблица 1-1.

Режим	Тип	Размеры	Адаптеры
0	текст,ч/б	40*25	CGA,EGA
1	текст 16 цветов	40*25	CGA,EGA
2	текст ч/б	80*25	CGA,EGA
3	текст 16 цветов	80*25	CGA,EGA
4	графика 4 цвета	320*200	CGA,EGA
5	графика 4 серых тона	320*200	CGA,EGA
6	графика ч/б	640*200	CGA,EGA
7	текст ч/б	80*25	монохромный
8	графика 16 цветов	160*200	PCjr
9	графика 16 цветов	320*200	PCjr
10	графика 4 или 16 цв.	640*200	PCjr,EGA
13	графика 16 цветов	320*200	EGA
14	графика 16 цветов	640*200	EGA
15	графика 4 цвета	640*350	EGA

Символы, выводимые на экран, содержатся в некоторой

зарезервированной области памяти на адаптере дисплея. Адрес одноцветной информации B0000000H. И CGA, и EGA хранят информацию, начиная с B80000000H. (Они различны для того, чтобы позволить использовать раздельно текстовый и графический экран - но на практике это делается редко.) Хотя функции CGA и EGA различны в разных режимах, они одинаковы в режимах 2 и 3.

Каждый символ, выводимый на экран, требует два байта видео памяти. Первый байт содержит собственно символ, второй содержит аттрибуты экрана. Для цветного экрана байт аттрибутов интерпретируется так, как показано в таблице 1-2. Если у вас EGA или CGA, то по умолчанию принимается режим 3, и символы выводятся с байтом аттрибутов 7. Это значение включает три основных цвета, производя для символа белый цвет. Для переключения в инверсный режим небходимо выключить три основных бита и включить три фоновых бита, что дает значение 70H.

Одноцветный адаптер распознает биты мигания и интенсивности. К счастью, он разработан так, что интерпретирует аттрибут 7, как нормальный текст (белое на черном) и 70H как инверсное видео. Кстати, значение 1 дает подчеркнутые символы.

Таблица 1-2

Байт видеоадаптера

Бит	Двоичная величина	Значение при установке
0	1	голубой основной
1	2	зеленый основной
2	4	красный основной
3	8	малая интенсивность
4	16	голубой фоновый
5	32	зеленый фоновый
6	64	красный фоновый
7	128	мигающий символ

Каждый адаптер имеет по крайней мере в 4 раза больше памяти, чем необходимо для вывода текста в 80-символьном режиме. Для этого есть две причины. Во-первых, лишняя память нужна для графики, (конечно кроме одноцветного адаптера). Во-вторых это позволяет держать в памяти несколько экранов и потом просто переключаться между ними по мере необходимости. Каждая область памяти называется видеостраницей и эффект от переключения между видеостраницами впечатляющ. По умолчанию при инициализации ДОС используется страница 0, и виртуально все приложения используют страницу 0. По этой причине она используется и в этой главе. Однако, вы можете использовать и другие страницы, если захотите.

Имеется три способа доступа к видеоадаптеру. Первый это через прерывание ДОС, которое достаточно медленно для исчезающего меню. Второй - это через процедуры BIOS, которые быстрее, и на быстродействующих машинах, таких, как AT или PS/2 достаточно быстры, если меню невелики. Третий способ - это чтение и запись прямо в видеопамять, что происходит очень быстро, но требует большей работы от вас. Эта глава рассматривает два разных подхода в видео процедурах. Один использует BIOS, а другой прямой доступ к видеопамяти.

Доступ к экрану через BIOS

Из-за того, что исчезающие и иерархические меню должны сохранять информацию с того места экрана, на котором они расположены, и восстанавливать его после выбора, вы должны иметь процедуры, которые сохраняют и загружают часть экрана. Метод сохранения и восстановления части экрана, рассматриваемый в этом разделе связан с вызовами двух встроенных в BIOS функций, которые читают и записывают символы на экран.

Как вы знаете, вызовы BIOS могут быть очень медленными. Однако, они (более или менее) гарантируют работу на любом

компьютере, который имеет BIOS, совместимый с IBM, даже если аппаратура экрана другая. Позже в этой главе вы узнаете, как выполнять прямой доступ к видеопамяти на IBM PC и 100% совместимых машинах для того, чтобы увеличить скорость выполнения. Однако, использование прямого доступа к видеопамяти снижает в некоторой степени переносимость, так как требуется 100% совместимость компьютера с IBM PC. Программы меню, основанные на BIOS следует использовать в применениях, которые требуют большей мобильности.

Использование int86**()**

Вызовы BIOS используют программные прерывания. BIOS имеет несколько различных прерываний для разных целей. Одно из них мы будем использовать для доступа к экрану. Это прерывание 16 **(**10H**)**, которое используется для доступа к дисплею. **(**Если вы не знакомы с доступом к BIOS, то вы найдете хорошее описание в моей книге "Си: Полный справочник", Беркли, 1987**)**. Как и многие прерывания BIOS, прерывание 16 имеет несколько режимов, выбор которых выполняется по значению регистра AH. Если функция возвращает значение, то оно заносится в регистр AL. Однако, иногда для возвращения нескольких значений используются другие регистры. Для доступа к прерываниям BIOS вам придется использовать функцию Си int86**()**. **(**Некоторые компиляторы могут называть эту функцию другим именем, но MicroSoft C и Турбо Си называют ее int86**()**. Последующие рассуждения ориентированы на эти трансляторы, но вы можете их обобщить.

Функция int86**()** имеет следующую форму:

```
 int int86(num,inregs,outregs)
 int num; /* номер прерывания */
 union REGS *inregs; /* входные значения регистров */
 union REGS *outregs; /* выходные значения регистров */
```

Функция int86**()** возвращает значение регистра AX. Тип REGS описывается в заголовке DOS.H. Этот тип показан здесь так, как он определен в Турбо Си, однако, он аналогично определен в MisroSoft C и в других компиляторах.

```
struct WORDREGS {
 unsigned int ax, bx, cx, dx, si, di, cflag, flags;
};
struct BYTEREGS {
unsigned char al, ah, bl, bh, cl, ch, dl, dh;
};
union REGS {
struct WORDREGS x;
struct BYTEREGS h;
};
```

Как вы можете видеть, REGS - это объединение двух структур. Использование структуры WORDREGS позволяет рассматривать регистры ЦП как 16-битные числа. BYTREGS дает вам доступ к отдельным 8-битным регистрам. Например, для доступа к прерыванию 16, функции 5, вы должны использовать следующую последовательность.

```
union REGS in,out;
in.h.ah=5;
int86(16,&in,&out);
```

Сохранение части экрана.

Для сохранения содержимого экрана, должно быть прочитано и запомнено текущее значение каждой позиции экрана. Для считывания символа с определенной позиции экрана, используется прерывание 16, функция 8, которая возвращает символ и связанный с ним аттрибут текущей позиции курсора. Для считывания символа с определенного места экрана, вы должны иметь способ установки курсора. Хотя некоторые компиляторы Си поддерживают эту функцию, многие ее не имеют. Тем не менее показанная ниже функция **goto**_xy**()** может быть использована. Она использует прерывание 16,

функцию 2 с координатой столбца в DL и координатой ряда в DH. Видеостраница задается в BH (используется страница 0 по умолчанию).

```
/* установка курсора в x,y */
void goto_xy(x,y)
int x,y;
{
 union REGS r;
 r.h.ah=2; /* функция установки курсора */
 r.h.dl=y; /* координата колонки       */
 r.h.dh=x; /* координата строки        */
 r.h.bh=0; /* видео страница           */
 int86(0x10,&r,&r);
}
```

Прерывание 16, функция 8 возвращает символ из текущей позиции курсора в AL и его атрибут в AH. Функция save_video(), показанная здесь, считывает часть экрана, сохраняет информацию в буфер, и очищает эту часть экрана.

```
/* сохранение части экрана */
void save_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
 union REGS r;
 register int i,j;
 for(i=starty;i<endy;i++)
   for(j=startx;j<endx;j++) {
     goto_xy(j,i);
     r.h.ah=8; /* функция чтения символа   */
     r.h.bh=0; /* видео страница           */
     *buf_ptr++ = int86(0x10,&r,&r);
     putchar(' '); /* очистка экрана */
   }
}
```

Первые четыре параметра save_video определяют координаты

верхнего левого и правого нижнего угла сохраняемой области. Параметр buf_ptr это целый указатель на область памяти, которая содержит информацию. Она должна быть достаточно большой, чтобы разместить всю информацию, считанную с экрана.

Программы в этой главе размещают буфер динамически, но вы можете использовать любую другую схему, если это важно для ваших приложений. Не забудьте, однако, что буфер должен существовать, до тех пор, пока экран не вернется в исходное состояние. Эта функция также чистит область, записывая пробел в каждой позиции.

Восстановление экрана

Восстановление экрана после сделанного выбора из меню, заключается просто в записи предварительно запомненной информации назад в видео память. Для того, чтобы сделать это, используйте прерывание 16, функцию 9, которая требует, чтобы символ был в AL, аттрибут в BL, видео страница в BH, а количество записываемых символов в CX (в нашем случае 1). Функция restore_video(), описанная здесь, помещает информацию из буфера, на который указывает buf_ptr, на экран, заданный начальными и конечными координатами X и Y.

```
/* восстановление части экрана */
void restore_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
```

```
 union REGS r;
 register int i,j;
 for(i=starty;i<endy;i++)
   for(j=startx;j<endx;j++) {
     goto_xy(j,i);
     r.h.ah=9; /* функция записи символа   */
     r.h.bh=0; /* видео страница           */
     r.x.cx=1; /* число повторений символа */
     r.h.al=*buf_ptr++; /* символ          */
     r.h.bl=*buf_ptr++; /* атрибут         */
     *buf_ptr++ = int86(0x10,&r,&r);
   }
}
```

Создание исчезающих меню

Функции, создающей исчезающее меню, должна быть передана некоторая информация. Во-первых, это список предоставляемых меню режимов. Поскольку в меню передаются высвечиваемые строки, то простейший путь передачи списка строк в функцию - помещение их в двумерный массив и передача указателя на массив. Как утверждалось ранее, значение меню может быть выбрано либо передвижением освещенной области на нужное поле и нажатием ВК или нажатием клавиши, указывающей на это поле. Для того, чтобы функция знала, какие клавиши "горячие" и что они обозначают, ей должны быть переданы их имена. Лучший путь для этого - передать строку, которая содержит символы "горячих" клавиш в том же порядке, что и строки меню.

Функция popup() должна также знать как много режимов в меню, и поэтому это число должно быть передано ей. Она должна также знать где расположить меню, то есть нужны координаты X и Y. Наконец, в некоторых ситуациях может быть желательным помещать меню в рамку, а в других - нет. Поэтому должно быть передано значение рамка включена/выключена. Для того, чтобы начать разработку функции popup() нам нужно описание :

```
/* высветить исчезающее меню и возвратить выбор */
int popup(menu,keys,count,x,y,border)
char *menu[];  /* текст меню */
char *keys;    /* горячие клавиши */
int count;     /* число режимов */
int x,y;       /* координаты левого верхнего угла */
int border;    /* если 0 то без рамки */
```

Функция popup() делает следующее :

```
# Сохраняет область экрана под меню
# Высвечивает рамку, если надо
# Высвечивает меню
# Получает ответ пользователя
# Восстанавливает экран в исходное состояние
```

Две из этих целей были достигнуты в save_video() и restore_video(), описанных в предыдущем разделе. Давайте рассмотрим как достигнуть три оставшиеся.

Высвечивание меню.

Для того, чтобы высветить меню, необходимо помнить, что popup получает указатель на массив указателей на строки. Для высвечивания отдельных строк вы просто индексируете указатель, как массив. Каждый элемент в массиве является указателем на соответствующий элемент меню. Следующая функция display_menu() высвечивает каждый элемент меню.

```
/* высвечивание меню на своем месте */
void display_menu(menu,x,y,count)
char *menu[];
int x,y,count;
{
```

```
 register int i;
for(i=0;i<count;i++,x++) {
  goto_xy(x,y);
  printf(menu[i]);
  }
}
```

Как вы можете видеть, эта функция получает указатель на массив строк, которые надо вывести, координаты начала меню и количество строк в меню.

В дальнейшем простейший способ создать двумерный массив символов, который содержит строки меню - это создание переменной в общем виде:

```
char *<имя меню> [] = {
"первая строка",
"вторая строка",
 .
 .
 .
"N-ая   строка" };
```

Это описание автоматически заставляет компилятор Си поместить строки в таблицу строк. Переменная указывает на первый символ первой строки в таблице. Например, это описание создает переменную fruit (фрукт), которая указывает на "Я" в "Яблоко".

```
char *fruit[] = {
 "Яблоко",
 "Апельсин",
 "Груша",
 "Грейпфрут",
 "Малина",
 "Клубника"
};
```


Высвечивание рамки

Если нужна рамка, то можно воспользоваться нижеприведенной программой для вывода рамки вокруг меню с заданными координатами левого верхнего и правого нижнего углов. Она использует символы, которые являются частью стандартного набора символов на машинах, совместимых с IBM. Если вы хотите, вы можете выбрать другие.

```
void draw_border(startx,starty,endx,endy)
int   startx,starty,endx,endy;
{
   register int i;
   for(i=startx+1;i<endx;i++) {
     goto_xy(i,starty);
     putchar(179);
     goto_xy(i,endy);
     putchar(179);
   }
   for(i=starty+1;i<endy;i++) {
     goto_xy(startx,i);
     putchar(196);
     goto_xy(endx,i);
     putchar(196);
   }
   goto_xy(startx,starty); putchar(218);
   goto_xy(startx,endy  ); putchar(191);
   goto_xy(endx  ,starty); putchar(192);
   goto_xy(endx  ,endy  ); putchar(217);
}
```

Ввод выбора пользователя

Как утверждалось, пользователь может вводить выбор одним из двух способов. Во-первых с помощью клавиш СТРЕЛКА ВНИЗ и СТРЕЛКА ВВЕРХ может переместить освещение на строку и нажать Ввод для ее выбора, (Обычно освещение строки выполняется в инверсном режиме.) Освещенную строку также можно передвигать пробелом. Второй способ это нажатие клавиши, связанной с выбором. Функция get_resp(), показанная здесь, достигает этих целей.

```
/* ввести выбор пользователя */
get_resp(x,y,count,menu,keys)
int x,y,count;
char *menu[];
char *keys;
{
 union inkey {
   char ch[2];
   int i;
 } c;
int arrow_choice=0,key_choice;
y++;
/* осветить первый выбор */
goto_xy(x,y);
write_video(x,y,menu[0],REV_VID);
for(;;) {
 while(!bioskey(1)); /* ждать нажатия */
 c.i=bioskey(0);
/* вернуть выбор в номальный режим */
goto_hy(arrow_choice,y);
write_video(x+arrow_choice,y,
        menu[arrow_choice],norm_vid);
if(c.ch[0]) { /* обычная клавиша */
 key_choice= is_in(keys,tolower(c.ch[0]));
if(key_choice) return key_choice-1;
switch(c.ch[0]) {
  case '\r' : return arrow_choice;
  case ' '  : arrow_choice++;
              break;
 case ESC   : return -1; /* выйти */
   }
}
else { /* специальная клавиша */
 switch(c.ch[1]) {
  case 72 : arrow_choice--; /* стрелка вниз */
            break;
  case 80 : arrow_choice++; /* стрелка вверх */
            break;
     }
  }
if(arrow_choice==count) arrow_choice=0;
if(arrow_choice<0) arrow_choice=count-1;
    /* подсветить выбранную опцию */
 goto_xy(x+arrow_choice,y);
 write_video(x+arrow_choice,y,menu[arrow_choice],REV_VID);
 }
}
```

Когда get_resp() начинает выполняться, освещается первое значение меню. Макроопределение REV_VID определяется везде, как 70H, а NORM_VID как 7H. Клавиша ESCAPE используется для окончания работы с меню. Значение ESC 27. После этого программа входит в цикл, ожидающий действий пользователя. Она использует функцию bioskey() для того, чтобы дождаться нажатия клавиши, а затем для считывания с этой клавиши. Функция bioskey() специфична для Турбо

Си. Если вы используете другой транслятор, вы можете использовать следующую версию функции.

```
/* эмуляция части функции bioskey Турбо Си */
bioskey(c)
int c;
{
 switch(c) {
  case 0: return get_key();
  case 1: return kbhit();
  }
}
/* чтение 16 битного скан кода клавиши */
get_key()
{
 union REGS r;
 r.h.ah=0;
 return int86(0x16, &r,&r);
```

Причина, по которой вы должны использовать bioskey() вместо get**char()** состоит в том, что программа должна иметь возможность считывать полный 16 битный скан код, генерируемый нажатием клавиши. Если нажата символьная клавиша, то символ помещается в младшие 8 бит, а старшие 8 бит равны 0. Однако, если нажата специальная клавиша, такая, как стрелка, младший байт равен 0, а старший содержит код позиции клавиши. Коды позиции для стрелки вверх и стрелки вниз равны 72 и 80 соответственно. Такие функции, как get**char()**, возвращают только код символа, и поэтому необходимо получить прямо скан код помимо них.

Каждый раз при нажатии стрелки освещенная опция переходит в нормальное изображение, а следующая - освещается. Нажатие стрелки вниз, в момент освещения крайней нижней позиции, означает возвращение к первому значению. То же самое, но наоборот происходит когда нажимается стрелка вверх при освещенной первой строке.

Функция write_video() используется функцией get_resp() для записи строки на дисплей в позиции X,Y с определенным атрибутом. Write_video(), показанная здесь, используется для вывода строки меню в инверсном режиме, если она освещена, или в нормальном, если она не освещена.

```
/* вывод строки с определенным атрибутом */
void write_video(x,y,p,attrib)
int x,y;
char *p;
int attrib;
{
 union REGS r;
 register int i,j;
 for(i=y; *p; i++) {
  goto_xy(x,i);
     r.h.ah=9; /* функция записи символа   */
     r.h.bh=0; /* видео страница           */
     r.x.cx=1; /* число повторений символа */
     r.h.al=*p++; /* символ          */
     r.h.bl=attrib; /* атрибут         */
     int86(0x10,&r,&r);
  }
```

Функция is_in() возвращает позицию "горячей" клавиши в строке. Если пользователь нажал не ключевую клавишу, то is_in возвращает 0.

```
is_in(s,c)
char *s,c;
{
 register int i;
 for(i=0; *s; i++)
    if(*s++ == c) return i+1;
```

```
 return 0;
}
```

Функция popup()

А теперь, когда все части созданы, функция popup может быть записана, как это показано здесь.

```
/* вывести исчезающее меню и вернуть выбор
   возвращает -2, если меню не может быть создано
   возвращает -1, если пользователь нажал клавишу ESC
   в остальных случаях она возвращает номер выбранной
   альтернативы, начиная с 0 */
int popup(menu,keys,count,x,y,border)
char *menu[];  /* текст меню */
char *keys;    /* горячие клавиши */
int count;     /* число альтернатив */
int x,y;       /* координаты левого верхнего угла */
int border;    /* если 0 то без рамки */
{
 register int i,len;
 int endx endy choice;
 unsigned int *p;
 if((x>24)||(x<0)||(y>79)||(y<0)) {
  printf(" выход за пределы экрана");
  return -2;
 }
/* вычисление размеров */
len=0;
for(i=0;i<count;i++)
 if(strlen(menu[i]) > len) len=strlen(menu[i]);
endy=len+2+y;
endx=count+1+x;
if((endx+1>24) || (endy+1>79)) {
  printf(" выход за пределы экрана");
  return -2;
 }
/* размещение памяти для видео буфера */
p=(unsigned int *)malloc((endx-x+1)*(endy-y+1));
 if(!p) exit(1); /* Вы можете здесь сами обработать ошибку */
/* сохранение части экрана */
void save_video(startx,endx,starty,endy,p);
if(border) draw_border(x,y,endx,endy);
/* высвечивание меню на своем месте */
void display_menu(menu,x,y,count);
/* ввести выбор пользователя */
choice=get_resp(x,y,count,menu,keys)
/* восстановление части экрана */
void restore_video(startx,endx,starty,endy,p);
free(p);
return choice;
}
```

Как вы можете видеть, popup() проверяет выход за пределы экрана и слишком большой размер меню. Она возвращает -2, если возникла одна из этих ситуаций. Из-за того, что get_resp() возвращает -1, при нажатии клавиши ESC, возвращение этого значения функцией popup следует рассматривать как "уничтожение" меню. Если пользователь сделал выбор, то возвращаемое значение будет в пределах от 0 до count-1 с соответствием первому значению меню 0. Как уже указывалось, popup() использует динамическое размещение памяти для обеспечения временной памяти для информации об экране. Обычно это лучший подход, но вы можете свободно изменить его, если это важно для вашего приложения.

Общий обзор

Простая программа, показанная здесь, использует все программы, разработанные для использования исчезающих меню. Вы не видели только функции cls(), которая очищает экран. Некоторые трансляторы Си не имеют функции для этого, и если это так, то вы не можете использовать следующую программу (в чистом виде).

```
/* процедура исчезающего меню для работы в текстовом режиме */
#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#define BORDER 1
#define ESC 27
#define REV_VID 0x70
#define NORM_VID  7
void save_video(),restore_video();
void goto_xy(),cls(),write_video();
void display_menu(),draw_border();
char *fruit[] = {
 "Яблоко",
 "Апельсин",
 "Груша",
 "Грейпфрут",
 "Малина",
 "Клубника"
};
char *color[]={
 "Красный",
 "Желтый",
 "Оранжевый",
 "Зеленый"
};
char *apple_type[] = {
 "Красный деликатес",
 "Джонатан",
 "Белый налив",
 "Антоновка"
};
main()
{
 int i;
cls();
goto_xy(0,0);
for(i=0;i<25;i++)
 printf("Это тест исчезающего меню\n");


popup(fruit,"яагрмк",6,1,3,BORDER);
popup(color,"кжоз",4,5,10,BORDER);
popup(apple_type,"кдба",4,10,18,BORDER);
}
/* вывести исчезающее меню и вернуть выбор
   возвращает -2, если меню не может быть создано
   возвращает -1, если пользователь нажал клавишу ESC
   в остальных случаях она возвращает номер выбранной
   альтернативы, начиная с 0 */
int popup(menu,keys,count,x,y,border)
char *menu[];  /* текст меню */
char *keys;    /* горячие клавиши */
int count;     /* число альтернатив */
int x,y;       /* координаты левого верхнего угла */
int border;    /* если 0 то без рамки */
{
 register int i,len;
 int endx endy choice;
```

```
 unsigned char *p;
 if((x>24)||(x<0)||(y>79)||(y<0)) {
  printf(" выход за пределы экрана");
  return -2;
 }
/* вычисление размеров */
len=0;
for(i=0;i<count;i++)
 if(strlen(menu[i]) > len) len=strlen(menu[i]);
endy=len+2+y;
endx=count+1+x;
if((endx+1>24) || (endy+1>79)) {
  printf(" выход за пределы экрана");
  return -2;
 }
/* размещение памяти для видео буфера */
p=(unsigned int *)malloc((endx-x+1)*(endy-y+1));
 if(!p) exit(1); /* Вы можете здесь сами обработать ошибку */
/* сохранение части экрана */
void save_video(x,endx+1,y,endy+1,p);
if(border) draw_border(x,y,endx,endy);
/* высвечивание меню на своем месте */
void display_menu(menu,x,y,count);
/* ввести выбор пользователя */
choice=get_resp(x,y,count,menu,keys)
/* восстановление части экрана */
void restore_video(x,endx+1,y,endy+1,p);
free(p);
return choice;
}
/* высвечивание меню на своем месте */
void display_menu(menu,x,y,count)
char *menu[];
int x,y,count;
{
 register int i;
for(i=0;i<count;i++,x++) {
  goto_xy(x,y);
  printf(menu[i]);
  }
}
void draw_border(startx,starty,endx,endy)
int   startx,starty,endx,endy;
{
 register int i;
 for(i=startx+1;i<endx;i++) {
goto_xy(i,starty);
putchar(179);
goto_xy(i,endy);
putchar(179);
}
 for(i=starty+1;i<endy;i++) {
goto_xy(startx,i);
putchar(196);
goto_xy(endx,i);
putchar(196);
}
goto_xy(startx,starty); putchar(218);
goto_xy(startx,endy  ); putchar(191);
goto_xy(endx  ,starty); putchar(192);
goto_xy(endx  ,endy  ); putchar(217);
}
/* ввести выбор пользователя */
get_resp(x,y,count,menu,keys)
```

```
int x,y,count;
char *menu[];
char *keys;
{
 union inkey {
   char ch[2];
   int i;
 } c;
int arrow_choice=0,key_choice;
y++;
/* осветить первый выбор */
goto_xy(x,y);
write_video(x,y,menu[0],REV_VID);
for(;;) {
 while(!bioskey(1)); /* ждать нажатия */
 c.i=bioskey(0);
/* вернуть выбор в номальный режим */
goto_xy(arrow_choice,y);
write_video(x+arrow_choice,y,
        menu[arrow_choice],norm_vid);
if(c.ch[0]) { /* обычная клавиша */
 key_choice= is_in(keys,tolower(c.ch[0]));
if(key_choice) return key_choice-1;
switch(c.ch[0]) {
  case '\r' : return arrow_choice;
  case ' '  : arrow_choice++;
              break;
 case ESC   : return -1; /* выйти */
   }
}
else { /* специальная клавиша */
 switch(c.ch[1]) {
  case 72 : arrow_choice--; /* стрелка вниз */
            break;
  case 80 : arrow_choice++; /* стрелка вверх */
            break;
     }
  }
if(arrow_choice==count) arrow_choice=0;
if(arrow_choice<0) arrow_choice=count-1;
    /* подсветить выбранную опцию */
 goto_xy(x+arrow_choice,y);
 write_video(x+arrow_choice,y,menu[arrow_choice],REV_VID);
 }
}
/* вывод строки с определенным атрибутом */
void write_video(x,y,p,attrib)
int x,y;
char *p;
int attrib;
{
 union REGS r;
 register int i,j;
 for(i=y; *p; i++) {
  goto_xy(x,i);
     r.h.ah=9; /* функция записи символа   */
     r.h.bh=0; /* видео страница           */
     r.x.cx=1; /* число повторений символа */
     r.h.al=*p++; /* символ          */
     r.h.bl=attrib; /* атрибут         */
     int86(0x10,&r,&r);
  }
}
/* сохранение части экрана */
```

```
void save_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
 union REGS r;
 register int i,j;
 for(i=starty;i<endy;i++)
   for(j=startx;j<endx;j++) {
     goto_xy(j,i);
     r.h.ah=8; /* функция чтения символа   */
     r.h.bh=0; /* видео страница           */
     *buf_ptr++ = int86(0x10,&r,&r);
     putchar(' '); /* очистка экрана */
   }
}
/* восстановление части экрана */
void restore_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
 union REGS r;
 register int i,j;
 for(i=starty;i<endy;i++)
   for(j=startx;j<endx;j++) {
     goto_xy(j,i);
     r.h.ah=9; /* функция записи символа   */
     r.h.bh=0; /* видео страница           */
     r.x.cx=1; /* число повторений символа */
     r.h.al=*buf_ptr++; /* символ          */
     r.h.bl=*buf_ptr++; /* атрибут         */
     int86(0x10,&r,&r);
   }
}
/* очистка экрана */
void cls()
{
 union REGS r;
 r.h.ah=6; /* код прокрутки экрана */
 r.h.al=0; /* код очистки экрана */
 r.h.ch=0; /* начальная строка */
 r.h.cl=0; /* начальная колонка */
 r.h.dh=24; /* конечная строка */
 r.h.dl=79; /* конечная колонка */
 r.h.bh=7; /* очистка пустой строки */
 int86(0x10,&r,&r);
}
```

Вводите эту программу в ваш компьютер и запускаете ее. В ходе ее выполнения каждое меню будет высвечено и исчезнет. (В этой программе все ответы теряются, но реальное применение будет, конечно, их обрабатывать.) Даже если ваш компьютер очень быстрый, вы возможно заметите, что исчезновение и появление меню требуют определенной задержки. Единственный путь решения этой проблемы - читать и писать символы прямо в видео память, что и обсуждается в следующем разделе. Еще раз отметим, что единственное важное достоинство использование BIOS в том, что такие меню работают на любом компьютере, который поддерживает BIOS, совместимый с IBM, даже если компьютер не 100% совместимый.

Прямой доступ к видео памяти

Для создания меню, которые действительно "исчезают" вы должны миновать вызовы функций BIOS и прямо обращаться к видео памяти. Это позволяет высвечивать символы с молниеносной быстротой. При прямой записи и чтении из видео памяти вы можете

использовать исчезающие меню в реальном времени!

Чтение и запись в видео память требует использования ДАЛЬНИХ указателей. Если ваш компилятор не поддерживает дальних указателей, то вы не имеете прямого доступа к видео памяти. Дальние указатели могут быть поддерживаемы транслятором Си одним из двух способов. Первый - использование ключевого слова **far**, используемого в большинстве компиляторов. Они позволяют определять указатель, как дальний. Другой способ - использование большой модели памяти, в которой все указатели по умолчанию дальние. Программы, используемые в этой главе используют описатель **far**. Если вы хотите, вы можете просто удалить его и скомпилировать программу, используя транслятор с большой моделью памяти.

Определение расположения видео памяти

Одноцветный адаптер использует для видео памяти адрес B0000000H, а все остальные - B8000000H. Для того, чтобы программы с меню работали правильно с каждым адаптером, они должны знать, какой адаптер имеет система. К счастью, для этого существует простой способ. Прерывание BIOS 16, функция 15 возвращает текущий видео режим. Как упоминалось раньше, программы, разработанные в этой главе, требуют режима 2, 3 или 7. Адаптеры CGA и EGA могут использовать режим 2 и 3, но не режим 7. Только одноцветный адаптер использует этот режим. Таким образом, если текущий видео режим 7, то используется одноцветный адаптер, в остальных случаях это EGA или CGA. Для наших задач, в текстовом режиме EGA и CGA одинаковы и поэтому все равно, какой из адаптеров у системы. Таким образом функция popup() должна поверить какой из адаптеров у системы и присвоить глобальной переменной указатель на соответствующий адрес. Этот фрагмент программы позволяет сделать это.

```
vmode = video_mode();
 if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
   printf(" должен быть 80 символьный текстовый режим");
   exit(1);
}
 /* присвоить соответствующий адрес видео памяти */
if(vmode==7) vid_mem=(char far *)0xB0000000;
else   vid_mem=(char far *)0xB8000000;
```

Функция video_mode() возвращает текущий видео режим, и переменную vid_mem, объявленную везде как **char far**.

Изменение save_video() и restore_video()

Как только переменной vid_mem присвоен соответствующий адрес, появляется простой способ использовать ее для чтения и записи символов в видео память. Запомните, видео память требует двух байтов для каждого символа, один для символа, а другой для атрибута. Из-за того, что символьный байт первый, а атрибутный - второй, то каждой строке экрана требуется 160 байт. Для того, чтобы определить адрес отдельного символа вы должны использовать формулу:

```
 адрес = адрес_адаптера + X*160 + Y*2
```

Функции save_video() и restore_video() при использовании прямого доступа к видео памяти выглядят следующим образом.

```
void save_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
 register int i,j;
 char far *v, far *t;
 v=vid_mem;
 for(i=starty;i<endy;i++)
```

```
   for(j=startx;j<endx;j++) {
      t = v + (j*160) + i*2;  /* вычисляем адрес */
      *buf_ptr++ = *t++;      /* чтение символа */
      *buf_ptr++ = *t;        /* чтение атрибута */
      *(t-1) = ' ';           /* очистка окна */
    }
}
/* восстановление части экрана */
void restore_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
 register int i,j;
 char far *v, far *t;
 v=vid_mem;
 t=v;
 for(i=starty;i<endy;i++)
   for(j=startx;j<endx;j++) {
     v = t;
     v += (j*160) + i*2;   /* вычисляем адрес */
     *v++ = *buf_ptr++;    /* запись символа */
     *v = *buf_ptr++;      /* запись атрибута */
    }
}
```

Как вы видете, символы и атрибуты записываются или читаются с помощью использования указателей на видео память. Другие функции, которые читают и записывают символы преобразуются подобным образом.

Если весь доступ к дисплею делать прямым, требуется одна новая функция (показанная здесь). Функция write_**char()** записывает один символ в определенную позицию экрана с определенным атрибутом.

```
/* запись символа с определенным аттрибутом */
void write_char(x,y,ch,attrib)
int x,y;
char ch;
int attrib;
{
 register int i;
 char far *v;
 v=vid_mem;
 v += (x*160) +y*2;
 *v++ = ch; /* запись символа */
 *v = attrib; /* запись атрибута */
}
```

Полная версия исчезающих меню с прямым доступом приведена здесь с тем же простым тестовым примером. Введите его в в свой компьютер и сравните по производительности с версией, использующей BIOS. Как вы увидете, разница потрясающая. Кажется, что меню появляются и исчезают мгновенно.

```
/* Программа исчезающих меню для текстового режима
   с использованием прямого доступа к видео памяти */
#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#define BORDER 1
#define ESC 27
#define REV_VID 0x70
#define NORM_VID  7
void save_video(),restore_video();
void goto_xy(),cls(),write_video();
void display_menu(),draw_border();
```

```
char far *vid_mem;
char *fruit[] = {
 "Яблоко",
 "Апельсин",
 "Груша",
 "Грейпфрут",
 "Малина",
 "Клубника"
};
char *color[]={
 "Красный",
 "Желтый",
 "Оранжевый",
 "Зеленый"
};
char *apple_type[] = {
 "Красный деликатес",
 "Джонатан",
 "Белый налив",
 "Антоновка"
};
main()
{
 int i;
cls();
goto_xy(0,0);
for(i=0;i<25;i++)
 printf("Это тест исчезающего меню\n");
popup(fruit,"яагрмк",6,1,3,BORDER);
popup(color,"кжоз",4,5,10,BORDER);
popup(apple_type,"кдба",4,10,18,BORDER);
}
/* вывести исчезающее меню и вернуть выбор
   возвращает -2, если меню не может быть создано
   возвращает -1, если пользователь нажал клавишу ESC
   в остальных случаях она возвращает номер выбранного
   режима, начиная с 0 */
int popup(menu,keys,count,x,y,border)
char *menu[];  /* текст меню */
char *keys;    /* горячие клавиши */
int count;     /* число режимов */
int x,y;       /* координаты левого верхнего угла */
int border;    /* если 0 то без рамки */
{
 register int i,len;
 int endx endy choice;
 unsigned char *p;
 if((x>24)||(x<0)||(y>79)||(y<0)) {
  printf(" выход за пределы экрана");
  return -2;
 }
/* вычисление размеров */
len=0;
for(i=0;i<count;i++)
 if(strlen(menu[i]) > len) len=strlen(menu[i]);
endy=len+2+y;
endx=count+1+x;
if((endx+1>24) || (endy+1>79)) {
  printf(" выход за пределы экрана");
  return -2;
 }
vmode = video_mode();
 if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
   printf(" должен быть 80 символьный текстовый режим");
```

```
   exit(1);
}
 /* присвоить соответствующий адрес видео памяти */
if(vmode==7) vid_mem=(char far *)0xB0000000;
else   vid_mem=(char far *)0xB8000000;
/* размещение памяти для видео буфера */
p=(unsigned int *)malloc((endx-x+1)*(endy-y+1));
 if(!p) exit(1); /* Вы можете здесь сами обработать ошибку */
/* сохранение части экрана */
void save_video(x,endx+1,y,endy+1,p);
if(border) draw_border(x,y,endx,endy);
/* высвечивание меню на своем месте */
void display_menu(menu,x,y,count);
/* ввести выбор пользователя */
choice=get_resp(x,y,count,menu,keys)
/* восстановление части экрана */
void restore_video(x,endx+1,y,endy+1,p);
free(p);
return choice;
}
/* высвечивание меню на своем месте */
void display_menu(menu,x,y,count)
char *menu[];
int x,y,count;
{
 register int i;
for(i=0;i<count;i++,x++) {
  write_string(x,y,menu[i],NORM_VID);
  }
}
void draw_border(startx,starty,endx,endy)
int   startx,starty,endx,endy;
{
 register int i;
 char far *v,far *t;
 v=vid_mem;
 t=v;
 for(i=startx+1;i<endx;i++) {
 v += (i*160) + starty*2;
 *v++ = 179;
 *v = NORM_VID;
 v=t;
 v += (i*160) + endy*2;
 *v++ = 179;
 *v = NORM_VID;
 v=t;
}
 for(i=starty+1;i<endy;i++) {
 v += (startx*160) + i*2;
 *v++ = 196;
 *v = NORM_VID;
 v=t;
 v += (endx*160) + i*2;
 *v++ = 196;
 *v = NORM_VID;
 v=t;
}
write_char(startx,starty,218,NORM_VID);
write_char(startx,endy  ,191,NORM_VID);
write_char(endx  ,starty,192,NORM_VID);
write_char(endx  ,endy  ,217,NORM_VID);
goto_xy(startx,endy  ); putchar(191);
goto_xy(endx  ,starty); putchar(192);
goto_xy(endx  ,endy  ); putchar(217);
```

```
}
/* ввести выбор пользователя */
get_resp(x,y,count,menu,keys)
int x,y,count;
char *menu[];
char *keys;
{
 union inkey {
   char ch[2];
   int i;
 } c;
int arrow_choice=0,key_choice;
y++;
/* осветить первый выбор */
goto_xy(x,y);
write_string(x,y,menu[0],REV_VID);
for(;;) {
 while(!bioskey(1)); /* ждать нажатия */
 c.i=bioskey(0);
/* вернуть выбор в номальный режим */
goto_xy(arrow_choice,y);
write_string(x+arrow_choice,y,
        menu[arrow_choice],norm_vid);
if(c.ch[0]) { /* обычная клавиша */
 key_choice= is_in(keys,tolower(c.ch[0]));
if(key_choice) return key_choice-1;
switch(c.ch[0]) {
  case '\r' : return arrow_choice;
  case ' '  : arrow_choice++;
              break;
 case ESC   : return -1; /* выйти */
   }
}
else { /* специальная клавиша */
 switch(c.ch[1]) {
  case 72 : arrow_choice--; /* стрелка вниз */
            break;
  case 80 : arrow_choice++; /* стрелка вверх */
            break;
     }
  }
if(arrow_choice==count) arrow_choice=0;
if(arrow_choice<0) arrow_choice=count-1;
    /* подсветить выбранную опцию */
 goto_xy(x+arrow_choice,y);
 write_string(x+arrow_choice,y,menu[arrow_choice],REV_VID);
 }
}
/* вывод строки с определенным атрибутом */
void write_string(x,y,p,attrib)
int x,y;
char *p;
int attrib;
{
 register int i,j;
 char far *v;
 v=vid_mem;
 v += (x*160) + y*2;
 for(i=y; *p; i++) {
   *v++ =*p++; /* запись символа */
   *v++ =attrib; /* запись атрибута */
  }
}
/* запись символа с определенным аттрибутом */
```

```
void write_char(x,y,ch,attrib)
int x,y;
char ch;
int attrib;
{
 register int i;
 char far *v;
 v=vid_mem;
 v += (x*160) +y*2;
 *v++ = ch; /* запись символа */
 *v = attrib; /* запись атрибута */
}
/* сохранение части экрана с использованием
   прямого доступа к видео памяти */
void save_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
 register int i,j;
 char far *v, far *t;
 v=vid_mem;
 for(i=starty;i<endy;i++)
   for(j=startx;j<endx;j++) {
      t = v + (j*160) + i*2;  /* вычисляем адрес */
      *buf_ptr++ = *t++;      /* чтение символа */
      *buf_ptr++ = *t;        /* чтение атрибута */
      *(t-1) = ' ';           /* очистка окна */
    }
}
/* восстановление части экрана с использованием
   прямого доступа к видео памяти */
void restore_video(startx,endx,starty,endy,buf_ptr)
int   startx,endx,starty,endy;
unsigned int *buf_ptr;
{
 register int i,j;
 char far *v, far *t;
 v=vid_mem;
 t=v;
 for(i=starty;i<endy;i++)
   for(j=startx;j<endx;j++) {
     v = t;
     v += (j*160) + i*2;   /* вычисляем адрес */
     *v++ = *buf_ptr++;    /* запись символа */
     *v = *buf_ptr++;      /* запись атрибута */
    }
}
/* очистка экрана */
void cls()
{
 union REGS r;
 r.h.ah=6; /* код прокрутки экрана */
 r.h.al=0; /* код очистки экрана */
 r.h.ch=0; /* начальная строка */
 r.h.cl=0; /* начальная колонка */
 r.h.dh=24; /* конечная строка */
 r.h.dl=79; /* конечная колонка */
 r.h.bh=7; /* очистка пустой строки */
 int86(0x10,&r,&r);
}
/* установка курсора в x,y */
void goto_xy(x,y)
int x,y;
{
```

```
 union REGS r;
 r.h.ah=2; /* функция установки курсора */
 r.h.dl=y; /* координата колонки       */
 r.h.dh=x; /* координата строки        */
 r.h.bh=0; /* видео страница           */
 int86(0x10,&r,&r);
}
/*  запрос текущего видео режима */
video_mode()
{
 union REGS r;
 r.h.ah = 15;   /* получить режим */
 return int86(0x10,&r,&r) & 255;
}
is_in(s,c)
char *s,c;
{
 register int i;
 for(i=0; *s; i++)
    if(*s++ == c) return i+1;
 return 0;
}
```

Создание иерархических окон

Иерархические окна фундаментально отличаются от простых исчезающих меню тем, что два или более исчезающих меню могут быть активными одновременно. Вообще иерархические меню позволяют пользователю выбирать режимы непосредственно из режимов и используются для поддержки системы меню. В отличие от функции popup(), которая сохраняет экран, высвечивает меню, и восстанавливает экран, функция pulldown(), разработанная в этом разделе только сохраняет экран (если это нужно), высвечивает меню и возвращает выбор пользователя. Восстановление экрана обрабатывается как отдельная задача в любом месте программы. Перед тем как вы сможете создать иерархическое меню, вы должны изменить свое представление о меню.

Фреймы меню

Центральным понятием для создания иерархического, многоуровневого меню является фрейм меню. В сущности, программы иерархических меню требуют чтобы каждое меню имело свой фрейм ссылок, определенных до того, как программа, которая испольует меню, начала выполняться. Каждое меню активизируется по номеру его фрейма и необходимая информация загружается по мере необходимости для различных функций, поддерживающих меню.

Лучший способ для поддержки фреймов меню - создать массив структур, которые содержат информацию, относящуюся к меню. Эта структура определяется так, как описано здесь:

```
 struct menu_frame {
   int startx,endx,starty,endy;
   unsigned char *p; /* указатель на информацию экрана */
   char **menu;      /* указатель на строки меню */
   int border;       /* рамка включено/выключено */
   int count;        /* число альтернатив */
   int astive;       /* активно ли меню сейчас */
 } frame[MAX_FRAME];
```

где MAX_FRAME - макроконстанта, которая определяет как много меню вы можете иметь. Только одна дополнительная информация требуется для иерархических меню, которая не нужна для исчезающих меню - флаг активности. Флаг используется в качестве сигнала, что меню уже на экране и предупреждает перезаписывание информации с экрана.

Создание фрейма меню

Перед использованием меню для этого должен быть создан фрейм. Функция make_menu(), показанная здесь, создает фрейм меню.

```
 /* создание фрейма иерархического меню.
    1 если фрейм может быть создан
    в противном случае 0 */
make_menu(num,menu,keys,count,x,y,border)
int num;   /* номер меню */
char *menu;   /* текст меню */
char *keys;   /* горячие клавиши */
int count;    /* число альтернатив */
int x,y;      /* левый верхний угол */
int border;   /* рамка */
{
 register int i,len;
 int endx,endy,choice,vmode;
 unsigned char *p;
 if(num>MAX_FRAME) {
  printf("Слишком много меню");
  return 0;
}
 if((x>24)||(x<0)||(y>79)||(y<0)) {
  printf(" выход за пределы экрана");
  return 0;
 }
/* вычисление размеров */
len=0;
for(i=0;i<count;i++)
 if(strlen(menu[i]) > len) len=strlen(menu[i]);
endy=len+2+y;
endx=count+1+x;
if((endx+1>24) || (endy+1>79)) {
  printf(" выход за пределы экрана");
  return 0;
 }
/* размещение памяти для видео буфера */
p=(unsigned int *)malloc((endx-x+1)*(endy-y+1));
 if(!p) exit(1); /* Вы можете здесь сами обработать ошибку */
/* создание фрейма */
frame[num].startx=x;
frame[num].endx=endx;
frame[num].starty=y;
frame[num].endy=endy;
frame[num].p = p;
frame[num].menu = (char **) menu;
frame[num].border = border;
frame[num].keys = keys;

frame[num].count = count;
frame[num].active =0;
return 1;
}
```

Вы вызываете make_menu с теми же аргументами, какие используются в popup() кроме номера меню, который должен быть определен в первом аргументе. Этот номер используется для идентификации меню.

Функция pulldown()

Функция pulldown() показана здесь:

```
/* Высветить меню и получить выбор
   возвращает -1, если пользователь нажал клавишу ESC
```

```
   в остальных случаях номер альтернативы, начиная с 0 */
int pulldown(num)
int num; /* номер фрейма */
{
 int vmode,choice;
 vmode=video_mode();
 if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
   printf(" должен быть 80 символьный текстовый режим");
   exit(1);
}
 /* присвоить соответствующий адрес видео памяти */
if(vmode==7) vid_mem=(char far *)0xB0000000;
else   vid_mem=(char far *)0xB8000000;
/* узнать активность окна */
if(!frame[num].active) { /* не активно */
    save_video(num);
    frame[num].active= 1; /* установить признак активности */
}
if( frame[num].border) draw_worder(num);
display_menu(num);  /* высветить меню */
return get_resp(num);  /* возвратить выбор */
}
```

Для использования pulldown() просто передайте номер того меню, которое вы хотите высветить. Однако вы должны помнить о восстановлении окна, используя restore_video() везде в своей программе. Запомните, что основное отличие иерархических меню в том, что они позволяют двум или более меню оставаться на экране одновременно и быть потенциально активными, по мере того как пользователь выбирает опции. Тем не менее вам не следует восстанавливать экран до того, как процесс выбора полностью завершен.

Обратите внимание, что часть экрана, используемая для меню, сохраняется только если признак активности равен 0. Так как иерархическое меню может быть вызвано повторно, экран не должен сохраняться много раз. (Другими словами при повторном вхождении будет сохранено само меню, записанное поверх первоначального содержимого экрана, которое уже сохранено.)

Восстановление экрана

Как и другие функции поддержки меню, измененная restore_video показанная здесь, преобразована для работы с фреймами. Поэтому функции restore_video() теперь передается только номер меню, что делает интерфейс более очевидным.

```
/* восстановление части экрана */
void restore_video(num)
int   num;
{
 register int i,j;
 char far *v, far *t;
 char *buf_ptr;
 buf_ptr=frame[num].p;
 v=vid_mem;
 t=v;
 for(i=frame[num].starty;i<frame[num].endy;i++)
   for(j=frame[num].startx;j<frame[num].endx;j++) {
     v = t;
     v += (j*160) + i*2;    /* вычисляем адрес */
     *v++ = *buf_ptr++;     /* запись символа */
     *v = *buf_ptr++;       /* запись атрибута */
    }
 frame[num].active= 0;
}
```

Простая программа, использующая процедуру pulldown

Все функции для иерархических меню показаны здесь вместе с простой программой-образцом и их можно прямо вводить в ваш компьютер.

```
/* процедура иерархического меню для текстового режима
   и простая программа-пример */
#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#define BORDER 1
#define ESC 27
#define REV_VID 0x70
#define NORM_VID  7
void save_video(),restore_video();
void goto_xy(),cls(),write_video();
void display_menu(),draw_border();
char far *vid_mem;
 struct menu_frame {
   int startx,endx,starty,endy;
   unsigned char *p; /* указатель на информацию экрана */
   char **menu;      /* указатель на строки меню */
   int border;       /* рамка включено/выключено */
   int count;        /* число альтернатив */
   int astive;       /* активно ли меню сейчас */
 } frame[MAX_FRAME];
char *fruit[] = {
 "Яблоко",
 "Апельсин",
 "Груша",
 "гРейпфрут",
 "Малина"
 "Клубника"
};
char *color[]={
 "Красный",
 "Желтый",
 "Оранжевый",
 "Зеленый"
};
char *apple_type[] = {
 "Красный деликатес",      ?
 "Джонатан",
 "Белый налив",
 "Антоновка"
};

char *grape_type[]= {
  "Конкорд",
  "кАнадский",
  "Томпсон",
  "кРасное пламя"
};
main()
{
 int i;
 cls();
 goto_xy(0,0);
/* во-первых создадим фреймы меню */
make_menu(0,fruit,"яагрмк",6,5,20,BORDER);
make_menu(1,color,"кжоз",4,9,28,BORDER);
make_menu(2,apple_type,"кдба",4,12,32,BORDER);
make_menu(3,grape_type,"катр",4,9,10,BORDER);
printf("Выберите фрукт:");
```

```
pd_driver(); /* запуск системы меню */
}
void pd_driver()
{
 int choice1,choice2,selection;
 /* активизация окон по мере надобности */
while((choice1=pulldown(0)) != -1) {
  switch ( choice1 ) {
    case 0 :  /* яблоко */
      while((choice2=pulldown(1)) != -1) {
        if(choice2 ==0) selection=pulldown(2);/*красное яблоко */
        restore_video(2);
      }
      restore_video(1);
      break;
    case 1 :
    case 2 : goto_xy(1,0);
      printf("неправильный выбор");
      break;
    case 3 : /* грейпфрут */
      selection=pulldown(3);
      restore_video(3);
      break;
    case 4 :
    case 5 : goto_xy(1,0);
      printf("неправильный выбор");
      break;
  }

 }
    restore_video(0);
}
/* Высветить меню и получить выбор */
int pulldown(num)
int num; /* номер фрейма */
{
 int vmode,choice;
 vmode=video_mode();
 if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
   printf(" должен быть 80 символьный текстовый режим");
   exit(1);
}
 /* присвоить соответствующий адрес видео памяти */
if(vmode==7) vid_mem=(char far *)0xB0000000;
else   vid_mem=(char far *)0xB8000000;
/* узнать активнсть окна */
if(!frame[num].active) { /* не активно */
    save_video(num);
    frame[num].active= 1; /* установить признак активности */
}
if( frame[num].border) draw_worder(num);

display_menu(num);  /* высветить меню */
return get_resp(num);  /* возвратить выбор */
}
 /* создание фрейма иерархического меню.
    1 если фрейм может быть создан
    в противном случае 0 */
make_menu(num,menu,keys,count,x,y,border)
int num;   /* номер меню */
char *menu;   /* текст меню */
char *keys;   /* горячие клавиши */
int count;    /* число альтернатив */
int x,y;      /* левый верхний угол */
```

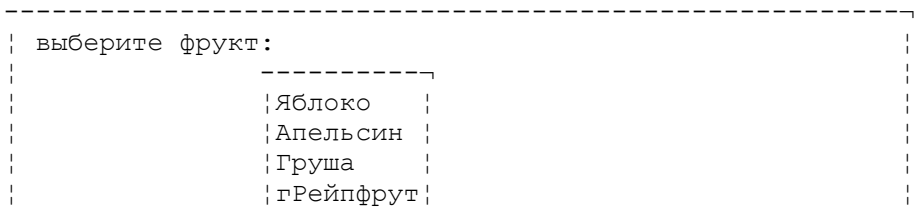
```
int border;   /* рамка */
{
 register int i,len;
 int endx,endy,choice,vmode;
 unsigned char *p;
 if(num>MAX_FRAME) {
  printf("Слишком много меню");
  return 0;
}
 if((x>24)||(x<0)||(y>79)||(y<0)) {
  printf(" выход за пределы экрана");

  return 0;
 }
/* вычисление размеров */
len=0;
for(i=0;i<count;i++)
 if(strlen(menu[i]) > len) len=strlen(menu[i]);
endy=len+2+y;
endx=count+1+x;
if((endx+1>24) || (endy+1>79)) {
  printf(" выход за пределы экрана");
  return 0;
 }
/* размещение памяти для видео буфера */
p=(unsigned int *)malloc((endx-x+1)*(endy-y+1));
 if(!p) exit(1); /* Вы можете здесь сами обработать ошибку */
/* создание фрейма */
frame[num].startx=x;
frame[num].endx=endx;
frame[num].starty=y;
frame[num].endy=endy;
frame[num].p = p;
frame[num].menu = (char **) menu;
frame[num].border = border;
frame[num].keys = keys;
frame[num].count = count;
frame[num].active =0;
return 1;
}
/* высвечивание меню на своем месте */
void display_menu(num)
 int num;
{
 char **m;
 register int i, x;
 x = frame[num].startx+1;
 m = frame[num].menu;
for(i=0;i<frame[num].count;i++,x++) {
  write_string(x,frame[num].starty+1,m[i],NORM_VID);
  }
}
void draw_border(num)
int  num ;
{
 register int i;
 char far *v,far *t;
 v=vid_mem;

 t=v;
 for(i=frame[num].startx+1;i<frame[num].endx;i++) {
 v += (i*160) + frame[num].starty*2;
 *v++ = 179;
 *v = NORM_VID;
```

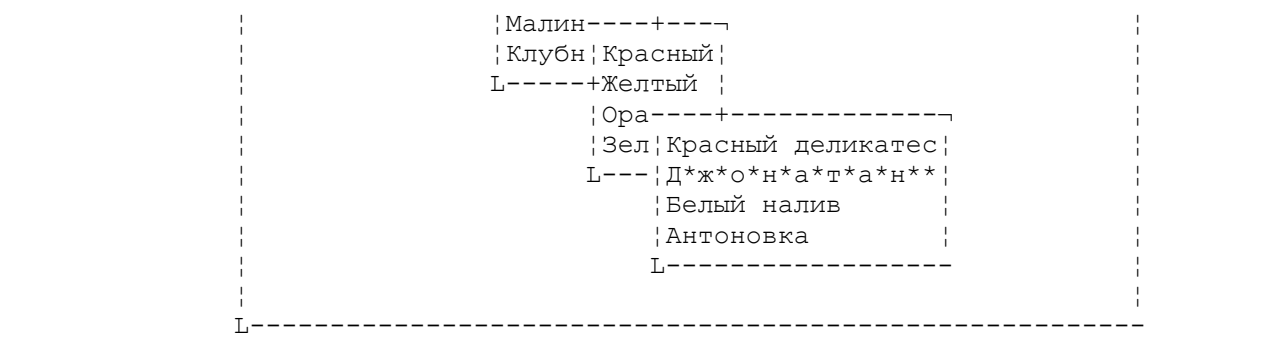
```
 v=t;
 v += (i*160) + frame[num].endy*2;
 *v++ = 179;
 *v = NORM_VID;
 v=t;
}
 for(i=frame[num].starty+1;i<frame[num].endy;i++) {
 v += (frame[num].startx*160) + i*2;
 *v++ = 196;
 *v = NORM_VID;
 v=t;
 v += (frame[num].endx*160) + i*2;
 *v++ = 196;
 *v = NORM_VID;
 v=t;
}
write_char(frame[num].startx,frame[num].starty,218,NORM_VID);
write_char(frame[num].startx,frame[num].endy  ,191,NORM_VID);
write_char(frame[num].endx  ,frame[num].starty,192,NORM_VID);
write_char(frame[num].endx  ,frame[num].endy  ,217,NORM_VID);
goto_xy(frame[num].startx,frame[num].endy  ); putchar(191);
goto_xy(frame[num].endx  ,frame[num].starty); putchar(192);
goto_xy(frame[num].endx  ,frame[num].endy  ); putchar(217);
}
/* ввести выбор пользователя */
get_resp(num)
int num;
{
 union inkey {
   char ch[2];
   int i;
 } c;
int arrow_choice=0,key_choice;
int x,y;
x=frame[num].startx+1;
y=frame[num].starty+1;
/* осветить первый выбор */
goto_xy(x,y);
write_string(x,y,frame[num].menu[0],REV_VID);
for(;;) {
 while(!bioskey(1)); /* ждать нажатия */
 c.i=bioskey(0);

/* вернуть выбор в номальный режим */
goto_xy(arrow_choice,y);
write_string(x+arrow_choice,y,
        frame[num].menu[arrow_choice],norm_vid);
if(c.ch[0]) { /* обычная клавиша */
 key_choice= is_in(frame[num].keys,tolower(c.ch[0]));
if(key_choice) return key_choice-1;
switch(c.ch[0]) {
  case '\r' : return arrow_choice;
  case ' '  : arrow_choice++;
              break;
 case ESC   : return -1; /* выйти */
   }
}
else { /* специальная клавиша */
 switch(c.ch[1]) {
  case 72 : arrow_choice--; /* стрелка вниз */
            break;
  case 80 : arrow_choice++; /* стрелка вверх */
            break;
     }
```

```
  }
if(arrow_choice==frame[num].count) arrow_choice=0;
if(arrow_choice<0) arrow_choice=frame[num].count-1;
    /* подсветить выбранную опцию */
 goto_xy(x+arrow_choice,y);
 write_string(x+arrow_choice,y,
              frame[num].menu[arrow_choice],REV_VID);
 }
}
/* вывод строки с определенным атрибутом */
void write_string(x,y,p,attrib)
int x,y;
char *p;
int attrib;
{
 register int i,j;
 char far *v;
 v=vid_mem;
 v += (x*160) + y*2;
 for(i=y; *p; i++) {
   *v++ =*p++; /* запись символа */
   *v++ =attrib; /* запись атрибута */
  }
}
/* запись символа с определенным аттрибутом */
void write_char(x,y,ch,attrib)
int x,y;
char ch;
int attrib;

{
 register int i;
 char far *v;
 v=vid_mem;
 v += (x*160) +y*2;
 *v++ = ch; /* запись символа */
 *v = attrib; /* запись атрибута */
}
/* сохранение части экрана с использованием
   прямого доступа к видео памяти */
void save_video(num)
 int num;
{
 register int i,j;
 char far *v, far *t;
 char *buf_ptr;
 buf_ptr=frame[num].p;
 v=vid_mem;
 for(i=frame[num].starty;i<frame[num].endy;i++)
   for(j=frame[num].startx;j<frame[num].endx;j++) {
      t = (v + (j*160) + i*2);  /* вычисляем адрес */
      *buf_ptr++ = *t++;      /* чтение символа */
      *buf_ptr++ = *t;        /* чтение атрибута */
      *(t-1) = ' ';           /* очистка окна */
    }
}
/* восстановление части экрана */
void restore_video(num)
int   num;
{
 register int i,j;
 char far *v, far *t;
 char *buf_ptr;
 buf_ptr=frame[num].p;
```

```
 v=vid_mem;
 t=v;
 for(i=frame[num].starty;i<frame[num].endy;i++)
   for(j=frame[num].startx;j<frame[num].endx;j++) {
     v = t;
     v += (j*160) + i*2;    /* вычисляем адрес */
     *v++ = *buf_ptr++;     /* запись символа */
     *v = *buf_ptr++;       /* запись атрибута */
    }
 frame[num].active= 0;
}
/* очистка экрана */
void cls()
{

 union REGS r;
 r.h.ah=6; /* код прокрутки экрана */
 r.h.al=0; /* код очистки экрана */
 r.h.ch=0; /* начальная строка */
 r.h.cl=0; /* начальная колонка */
 r.h.dh=24;/* конечная строка */
 r.h.dl=79;/* конечная колонка */
 r.h.bh=7; /* очистка пустой строки */
 int86(0x10,&r,&r);
}
/* установка курсора в x,y */
void goto_xy(x,y)
int x,y;
{
 union REGS r;
 r.h.ah=2; /* функция установки курсора */
 r.h.dl=y; /* координата колонки       */
 r.h.dh=x; /* координата строки        */
 r.h.bh=0; /* видео страница           */
 int86(0x10,&r,&r);
}
/*  запрос текущего видео режима */
video_mode()
{
 union REGS r;
 r.h.ah = 15;   /* получить режим */
 return int86(0x10,&r,&r) & 255;
}
is_in(s,c)
char *s,c;
{
 register int i;
 for(i=0; *s; i++)
    if(*s++ == c) return i+1;
 return 0;
}
```

В этом примере, если пользователь выберет "Яблоко", то он или она будет запрошен о цвете яблока; если выбран "Красный" цвет, то будет высвечен список красных сортов яблок. Если же будет выбран грейпфрут то пользователь будет запрошен о желаемом типе. Меню для выбора яблок показано на рисунке.





Посмотрите внимательно на функцию pd_driver**()**, которая следует за главной функцией main**()**. При использовании иерархических меню вы должны создавать функцию, которая управляет системой меню. Основа стратегии управляющей функции должна быть аналогична функции pd_driver**()** из этого примера. Не забывайте, что эта простая программа только иллюстрирует как активизировать меню. Ваша реальная прикладная программа будет обрабатывать выбранные режимы более разумным образом. Запомните, что для использования иерархических меню нужна следующая последовательность действий.

1. Создать меню, используя make_menu**()**.
2. Активизировать меню, используя pulldown**()**.
3. Восстановить экран, используя restore_video**()**, при выходе из каждого меню.

Добавочные опции

Процедуры меню, разработанные в этой главе, подходят в большинстве ситуаций. Однако, вы можете по желанию добавить некоторые из следующих возможностей:

Высвечивание заголовка меню
Линейное меню **(**все опции на одной строке**)**
Использование разных цветов для разных меню

ГЛАВА 2

ВСПЛЫВАЮЩИЕ ОКНА

Всплывающие окна могут придать вашей программе тот профессиональный вид, который не может быть достигнут другими средствами. Всплывающие окна создают впечатление, что вы, как программист, в совершенстве владеете экраном. А так как пользователь обычно судит о программе по ее пользовательскому интерфейсу, то это положительное впечатление распространится и на всю программу в целом.

Данная глава содержит описание полного набора функций для всплывающих окон, которые позволят вам создавать и использовать множественные окна.

Программы управления окнами используют функции прямого доступа к видеопамяти, представленные в Главе 1. Из-за того, что окна в большинстве случаев, имеют значительно больший размер, чем меню, то использование функций из ROM-BIOS просто невозможно, - даже на самых быстрых компьютерах.

Однако перед рассмотрением оконных функций очень важно правильно понять, что же такое всплывающие окна и как они используются.

Глава II
Теория всплывающих окон.

Всплывающее окно представляет собой часть экрана, используемую для специальных целей. Перед появлением окна текущее содержимое экрана сохраняется и лишь после этого производится отображение окна.

При завершении программы, использующей данное окно, это окно

удаляется, а первоначальное содержимое экрана восстанавливается. (Данный процесс аналогичен появлению всплывающих меню). Вполне возможно одновременное отображение на экране нескольких окон.

Хотя это и не обязательно, но все хорошие программы, работающие с окнами, позволяют интерактивно изменять размеры и позицию окна на экране. Следовательно, оконные функции допускают, что окно не всегда будет находиться в одном и том же месте и иметь один и тот же размер.

Разработка функций, управляющих окнами, является сложной задачей из-за необходимости обеспечения запрета для прикладной программы осуществлять вывод за границы окна. Поскольку размеры окна могут изменяться без "сообщения" об этом прикладной программе, то именно функции управления окнами, а не прикладная программа, должны предохранить от выхода за границы. Следовательно, все обычные функции Си, осуществляющие ввод/вывод на консоль (например, printf() и lets()) , не могут быть использованы и должны быть заменены на функции, ориентированные на ввод/вывод с использованием окон.

Теория использования окон крайне проста. Каждая отдельная задача программы использует свое собственное окно. При запуске задачи активируется и ее окно. При завершении работы задачи - ее окно удаляется. Если же задача прерывается, то, хотя ее работа приостанавливается, но ее окно не удаляется, а инициируемая прерыванием задача, просто создает свое окно поверх предыдущего. (Обычно те задачи, которые не используют окон, очищают экран. Это приводит к рассеиванию внимания пользователя. В то же время при использовании окон подобные прерывания выглядят как временные паузы).

Чтобы понять, как окна могут быть наиболее эффективно использованы, предположим, что вы разработали текстовый редактор, включающий ряд дополнительных функций, таких как "записная книжка", калькулятор с четырьмя математическими операциями и конвертер чисел из десятичного в шестнадцатиричное представление. Так как все эти функции в действительности не относятся к операциям редактирования текста, то их реализация тесно переплетается с концепцией всплывающих окон. Таким образом получается, что использование какой-либо из вспомогательных функций лишь приостанавливает основную задачу (редактирование), а не прерывает ее.

Глава II

Оконные структуры.

Правильная реализация всплывающих окон требует, чтобы все атрибуты, необходимые для описания их границ, были в любое время доступны всем оконным функциям. Для достижения этого мы будем использовать концепцию структуры, аналогичную той, которая использовалась при описании функций спускающихся меню. Однако структура окна содержит некоторую специфическую информацию. Ниже показан массив, используемый для хранения структур.

```
struct window_frame
  int startx, endx, starty, endy; /*позиция окна*/
  int curx, cury; /*текущая позиция курсора в окне*/
  unsigned chsr *p; /*указатель буфера*/
  char *header; /*имя окна*/
  int border; /*включение/выключение границ*/
  int active; /*на экране или невидимо*/
} frame [MAX_FRAME];
```

Переменные startx, starty, endx и endy хранят координаты верхнего левого и нижнего правого углов окна. Текущая позиция курсора в окне содержится в переменных curx и cury. Сохранение этих переменных осуществляется из-за того, что положение курсора может изменяться и вручную и путем использования оконных функций. Указатель p указывает на область памяти, хранящей первоначальное содержимое части экрана, занятой данным окном. Часто окно

снабжается заголовком, иденцифирующим содержимое окна. На этот заголовок и указывает header. Переменная border используется для определения необходимости вычерчивания границ вокруг окна. Переменная active установлена в "1", если в данный момент окно на экране, и в "0" - в противном случае.

С точки зрения программирования использование окон не составляет труда. Во-первых, вы создаете структуру окна, затем, когда появляется необходимость в окне, то при записи в него вы используете специальные оконно-ориентированные функции ввода/вывода. Когда же окно больше не нужно, вы деактивируете его.

Глава II
Создание структуры окна.

Ниже показана функция под названием make_window(), используемая для создания структуры окна.

```
    /* Создать рамку спускающегося окна.
     Возвратить 1, если рамка окна  может быть создана
     и 0 в противном случае */
    make_window(num, header,startx,starty,endx,endy,border)
    int num; /* номер окна */
    char *header; /* текст заголовка */
    int startx,starty; /* координаты X,Y левого верхнего угла */
    int endx,endy; /* координаты X,Y правого верхнего угла */
    int border; /* без бордюра, если 0 */
    {
      unsigned char *p;
      if(num>MAX_FRAME) {
        printf("Too many windows\n");
        return 0;
      }
     if((startx>24) || (startx<0) || (starty>78) || (starty<0)) {
        printf("range error");
        return 0;
      }
      if((endx>24) || (endy>79)) {
        printf("window won't fit");
        return 0;
      }
  /* отвести достаточное количество памяти */
  p=(unsigned char *) malloc(2*(endx-startx+1)*(endy-starty+1));
  if(!p) exit(1); /* перейти к высшему собственному обработчику
  ошибок */
      /* создать рамку */
      frame[num].startx = startx; frame[num].endx = endx;
      frame[num].starty = starty; frame[num].endy = endy;
      frame[num].p = p;
      frame[num].haeder = header;
      frame[num].border = border;
      frame[num].active = 0;
      frame[num].curx = 0; frame[num].cury = 0;
      return 1;
    }
```

Как вы можете видеть из описания функции, она требует передачи номера окна, структуру которого вы хотите создать, и всей остальной соответствующей информации. Ниже показан пример вызова make_window() для создания окна номер 0 с заголовком

Глава II

"Редактор [Esc для выхода]", с верхним левым углом в 0,0 и с нижним правым углом в 24,78 и имеющем границы.

```
 make_window (0, "Редактор [Ese для вывода]", 0,0,24,78, BORDER);
```

Отметим, что переменные, определяющие позицию курсора curx и cury устанавливаются 0. Это означает, что при первой активации окна, курсор будет установлен в его верхний левый угол. Функция

также управляет размещением окна на экране.

Глава II

Активирование и деактивирование окна.

Для активирования окна используется функция window **()**. Здесь num будет содержать номер структуры окна, которое вы хотите использовать.

```
/* Вывести на экран спускающееся окно */
void window(num)
int num; /* номер окна */
{
  int vmode, choice;
  int x, y;
  vmode = video_mode();
  if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
    printf("video must be in 80 column text mode");
    exit(1);
  }
  /* установить соответствующий адрес видеопамяти */
  if(vmode==7) vid_mem = (char far *) 0xB0000000;
  else vid_mem = (char far *) 0xB0000000;
  /* сделать окно активным */
  if(!frame[num].active) { /* используется не постоянно */
  save_video(num); /* сохранить текущее содержимое экрана */
  frame[num].active = 1; /* установить флаг активности */
  }
  if(!frame[num].border) draw border(num);
  display_header(num); /* вывести окно на экран */
  x = frame[num].startx + frame[num].curx + 1;
  y = frame[num].starty + frame[num].cury + 1;
  goto_xy(x,y);
}
```

Как вы можете видеть, данная функция очень похожа на функцию menu**()**, показанную в предыдущей главе. Переменная vid-mem является глобальным указателем типа **char far**.

Функция display_header**()**, показанная ниже, используется для отображения в центре верхней границы окна его заголовка. Если же заголовок не помещается в эту строку, то он не выводится.

```
/* вывести текст заголовка начиная с определенной
позиции */
void display_header(num)
int num;
{
  register int y,len;
```



```
  y = frame[num].starty;
  /* Вычислить начальную позицию относительно центра текста
  заголовка, если отрицательная, то текст не подходит */
  len = strlen(frame[num].header);
  len = (frame[num].endy - y - len) / 2;
  if(len<0) return; /* не выводить на экран */
  y = y +len;
  write_string(frame[num].startx, y,
               frame[num].header,NORM_VID);
}
```

Если вы хотите, чтобы заголовок выводился в инверсном изображении, то вместо NORM_VID **(**имеющем значение 7**)** подставьте REV_VID **(**со значением 70H**)**.

Для деактивации окна используется показанная ниже функция deactivate**()**, которой передается номер удаляемого окна.

```
/* Деактивировать окно и удалить его с экрана */
deactivate(num)
int num;
{
```

```
    /* установить курсор в левый верхний угол */
    frame[num].curx = 0;
    frame[num].cury = 0;
    restore_video(num);
  }
```

Как вы видите, функция устанавливает позицию курсора в 0,0. Однако, поскольку вам могут встретиться некоторые ситуации, в которых более желательно не устанавливать курсор в нулевую позицию, то вы можете изменить поведение этой функции, сообразуясь со своими намерениями.

Глава II
Оконные функции ввода/вывода.

Перед использованием окна необходимо разработать значительное число консольных оконно-ориентированных функций ввода/вывода. Чтобы понять, почему требуется так много функций, вспомните о том, сколько консольных функций ввода/вывода в стандартной библиотеке Си. Функции, представленные в данной главе, в действительности составляют лишь минимальный набор, необходимый для использования окон. Хотя они не включают в себя ориентированные на использование окон версии всех консольных Си-функций, но все же имеют большой объем. Как вы убедитесь, даже простейшие операции, такие как чтение символа с клавиатуры или вывод его на экран, реализуются программами большого объеема, поскольку необходимо отслеживать и сохранять текущую позицию курсора и не допускать выхода за границы окна. Помните, что для манипулирования экраном нельзя использовать никаких стандартных возможностей, предоставляемых DOS. Например, при необходимости выполнить "возврат каретки" это должно быть сделано самостоятельно, внутри функции; нельзя просто вызвать DOS для вывода соответствующей последовательности.

Для облегчения идентификации все оконные функции ввода/вывода начинаются со слова window. Кроме того, все эти функции в качестве своего первого аргумента принимают номер окна, к которому осуществляется доступ.

Глава II
Функция позиционирования курсора в окне.

Возможно, кому-то это покажется странным, но первая функция, в которой возникает потребность, это оконный эквивалент функции **goto_xy()**. Объяснение этому очень простое. Поскольку оконные функции ввода/вывода должны самостоятельно позиционировать курсор, то должен существовать некоторый способ установки курсора в нужную позицию. Функция window_xy(), представленная ниже, делает именно это. (Для представленных здесь программ работы с окнами позиция 0,0 соответствует левому верхнему углу окна).

```
  /* Установить курсор в определенную позицию окна.
     Возвратить 0 при выходе за границу и не ноль -
     в противном случае */
  window_xy(num, x, y,)
  int num, x, y;
  {
    if(x<0 || x+frame[num].startx>=frame[num].endx-1)
      return 0;
    if(x<0 || y+frame[num].starty>=frame[num].endy-1)
      return 0;
    frame[num].curx = x;
    frame[num].cury = y;
    goto_xy(frame[num].startx+x+1, frame[num].starty+y+1);
    return 1;
    }
```

Ключом к пониманию функции window_xy() является напоминание о том, что значения координат X,Y внутри окна остаются неизменными, независимо от расположения окна на экране. То есть

координаты X,Y вычисляются относительно окна, а не относительно экрана. Другими словами, если вы установите курсор в окне в позицию с координатами 2,2, то он всегда будет находиться во второй строке сверху и второй позиции относительно левого верхнего угла окна, независимо от положения окна на экране.

Фактически функция window_xy() производит преобразование координат курсора относительно окна в реальные координаты относительно экрана. Кроме этого, функция window_xy() не позволяет курсору выйти за пределы окна.

Глава II
Функция window_getche()

При работе с окнами вы не можете использовать функцию getche(), которая считывает введенный с клавиатуры символ и отображает его на экране, поскольку это может привести к выходу за пределы окна. Поэтому должна быть использована альтернативная оконная функция window_getche(). Эта функция считывает символ, который вводится в текущую позицию окна.

```
/*  Ввести с клавиатуры символ в  окно.  Возвратить
    полный 16-разрядный скан-код. */
 window_getche(num)
 int num;
 {
   union inkey {
     char ch[2];
     int i;
   } c;
   if(!frame[num].active) return 0; /* окно не активное */
   window_xy(num, frame[num].curx, frame[num].cury);
   c.i = bioskey(0); /* принять символ от клавиатуры */
   if(c.ch[0]) {
    switch(c.ch[0]) {
     case '\r':  /* нажата клавиша ENTER */
      break;
     case BKSP; /* возврат */
      break;
     default:
   if(frame[num].cury+frame[num].starty < frame[num].endy-1) {
     write_char(frame[num].startx = frame[num].curx=1,
       frame[num].starty+frame[num].cury+1,c.ch[0],NORM_VID);
         frame[num].cury++;
       }
     }
     if(frame[num].curx < 0) frame[num].curx = 0;
     if(frame[num].curx+frame[num].startx > frame[num].endx-2)
       frame[num].curx--;
     window_xy(num, frame[num].curx, frame[num].cury);
   }
   return c.i;
 }
```

В отличие от функции getche(), функция window_getche() возвращает полный 16-разрядный скан-код. Это означает, что вы имеете доступ как к стандартным кодам символов в младших восьми разрядах, так и к позиционным кодам символов в старших восьми разрядах. Если вам не нужны позиционные коды, то вы можете просто назначить возвращаемое функцией window_getche() значение

символьной переменной.

Функция работает следующим образом. Если окно не является активным (т.е. его нет на экране), функция возвращает 0. Поскольку код 0 не может соответствовать символу, введенному с клавиатуры, то ваша программа сможет обнаружить эту ситуацию. Затем курсор устанавливается в свою текущую позицию в окне, и

считывается код нажатой клавиши. Если это обычная клавиша, а не клавиша типа **RETURN** или BACKSPASE, инкрементируется текущее значение переменной Y, соответствующее положению курсора, и соответствующий символ выводится на экран. Если курсор находится на границе окна, значение Y декрементируется. Последнее обращение к функции window_xy**()** используется для того, чтобы переместить курсор в следующую позицию экрана.

Функция window_getche не позволит вам вводить символы за границей окна. Следует помнить, что все окна имеют границы, которые могут и не отображаться явно в виде линий. Если граница не отображается, это значит, что для этой цели используется символ пробела. Это имеет смысл, если окно отображается на фоне изображения, имеющего другой цвет. Даже если граница и не обозначена явно, то символы все равно не могут отображаться за пределами окна.

Как утверждается в разделе 1, функция bios_key**()** специфична для Турбо Си. Если вы используете другой компилятор Си, то вы можете использовать версию функции bios_key**()**, представленную в главе 1.

Глава II
Функция window_gets**()**

Для чтения строки, введенной в окно, можно использовать представленную ниже функцию window_gets**()**. Она не такая изощренная, как большинство функций gets**()**, но может служить для многих целей. Вы всегда можете расширить ее функциональные возможности, если пожелаете.

```
/* Считать строку из окна */
void window_gets(nums, s)
int num;
char *s;
{
  char ch, *temp;
  temp = s;
  for(;;) {
    ch = window_getche(num);
    switch(ch) {
      case '\r': /* нажата клавиша ENTER */
      *s='\0';
      return;
    case BKSP:   /* возврат */
      if(s>temp) {
        s--;
        frame[num].cury--;
        if(frame[num].cury<0) frame[num].cury = 0;
        window_xy(num, frame[num].curx, frame[num].cury);
        write_char(frame[num].startx+ frame[num].curx+1'
          frame[num].starty+frame[num].cury+1, ' ',NORM_VID);
      }
      break;
    default: *s = ch;
      s++;
    }
  }
}
```

При нажатии клавиши BACKSPASE, необходимо вернуть курсор на одну позицию влево, стереть записанный там символ и на его место записать пробел.

Глава II
Функция window_put**char()**

При выводе символа в окно необходимо проверять, является ли окно активным и не выходит ли символ за границу окна. После вывода символа курсор продвигается на одну позицию. Выполняет эти

действия представленная ниже функция window_put**char()**.

```
    /* Вывести символ в текущую позицию курсора в созданном окне.
    Возвратить  0, если окно не активное, и 1 - в противном
    случае */
    window_putchar(num, ch)
    int num;
    char ch;
    {
      register int x, y;
      char far *v;
      /* убедиться, что окно активное */
      if(!frame[num].active) return 0;
      x = frame[num].curx = frame[num].startx + 1;
      y = frame[num].cury = frame[num].starty + 1;
      v = vid_mem;
      v += (x*160) + y*2;  /* вычисляется адрес */
      if(y>=frame[num].endy) {
        return 1;
      }
      if(x>=frame[num].endx) {
        return 1;
      }
      if(ch=='\n') {  /* символ перехода на следующую строку */
        x++;
        y = frame[num].startx+1;
        v = vid_mem;
        v += (x*160) = y*2;  /* вычислить адрес */
        frame[num].curx++;   /* нарастить x */
        frame[num].cury = 0; /* нарастить y */
      }
      else {
        frame[num].cury++;
        *v++ = ch;           /* вывести символ */
        *v++ =NORM_VID;      /* нормальные атрибуты символа */
      }
      window_xy(num, frame[num].curx, frame[num].cury);
      return 1;
    }
```

Эта функция не занимается обнаружением ошибочнных ситуаций, когда символ выходит за границу окна. Смысл этого состоит в том, что размер окна может динамически изменяться и то сообщение,

которое помещалось в окно в следующий момент времени может не помещаться. В этом случае функция просто не отображает те символы, которые выходят за границу окна.

Обратите внимание, что нажатие клавиши возврата каретки требует перевода курсора к левой границе окна и на одну строку вниз, если это возможно.

Функция window_puts

Функция window_puts выводит заданную строку в активное окно, используя при этом функцю window_put**char()**.

```
 /* Вывести строку, начиная с текущей позиции курсора в окне.
                                                        /* 60 */
    Возвратить 0, если окно не активное и 1 в противном случае */
    window_puts(num, str)
    int num;
    char *str;
    {
      /* убедиться, что окно активное */
      if(!frame[num].active) return 0;
      for( ; *str; str++)
        window_putchar(num, *str);
```

```
      return 1;
    }
```


Дополнительные функции управления экраном.

При работе с окнами также используются следующие функции управления экраном:

Функция	Назначение
window cls()	очищает окно
window cleol()	очищает часть окна от текущей позиции до конца строки
window_upline()	перемещает курсор на одну строку вверх
window_downline()	перемещает курсор на одну строку вниз
window_bksp()	перемещает курсор на одну позицию влево

Эти функции представлены ниже. Следуя используемым в них общим принципам, вы можете создать свои функции управления экраном.

```
    /* Очистить окно */
    void window_cls(num)
    int num;
    {
      register int i,j;
      char far *v, far *t;
      v = vid_mem;
      t = v;
      for(i=frame[num].starty+1; i<frame[num].endy; i++)
        for(j=frame[num].startx+1; j<frame[num].endx; j++) {
          v = t;
          v += (j*160) + i*2;
          *v++ = ' ';     /* вывести пробел */
          *v =  NORM_VID; /* нормальные видеоатрибуты */
        }
        frame[num].curx = 0;
        frame[num].cury = 0;
      }
      /* очистить до конца строки */
      void window_cleol(num)
      int num;
      {
        register int i, x, y;
        x = frame[num].curx;
        y = frame[num].cury;
        window_xy(num, frame[num].curx, frame[num].cury);
        for(i=frame[num].cury; i<frame[num].endy-1; i++)
          window_putchar(num,' ');
        window_xy(num, x, y);
```



```
      }
/* Переместить курсор на одну строку вверх. Возвратить ненулевой
   код в случае успеха и 0 - в противном случае */
      window_upline(num)
      int num;
      {
        if(frame[num].curx>0) {
          frame[num].curx--;
          window_xy(num, frame[num].curx, frame[num].cury);
          return 1;
        }
        return 0;
      }
      window_downline(num)
      int num;
      {
```

```
      if(frame[num].curx<frame[num].endx-frame[num].startx-1) {
        frame[num].curx++;
        window_xy(num, frame[num].curx, frame[num].cury);
        return 1;
      }
      return 1;
    }
    /* стереть предыдущий символ */
    window_bksp(num)
    int num;
    {
      if(frame[num].cury>0) {
      frame[num].cury--;
      window_xy(num, frame[num].curx, frame[num].cury);
      window_putchar(num, ' ');
      frame[num].cury--;
      window_xy(num, frame[num].curx, frame[num].cury);
    }
  }
```

Глава II

Изменение размера и положения окна во время выполнения программы

Хотя функция make_window() и используется для установки начальных размеров и положения окна на экране, однако эти параметры могут динамически изменяться во время выполнения программы. Изменение одного или более параметров окна производится по командам, поступающим от пользователя. При этом текущее окно уничтожается и воссоздается уже с новыми параметрами. Представленные ниже программы size() и move() как раз и используются для изменения размеров и положения окна. Для изменения формы и положения окна используются клавиши со стрелками, а также клавиши HOME, END, PGDN и PGUP.

```
  /* Интерактивное изменение размера окна */
  void size(num)
  int num;
  {
    char ch;
    int x, y, startx, starty;
    /* активировать, если необходимо */
    if(!frame[num].active) window(num);
    startx = x = frame[num].startx;
    starty = y = frame[num].starty;
    window_xy(num, 0, 0);
    do {
      ch = get_special();
      switch(ch) {
        case 75:     /* влево */
          starty--;
          break;
        case 77:     /* вправо */
          starty++;
          break;
        case 72:     /* вверх */
          startx--;
          break;
        case 80:     /* вниз */
          startx++;
          break;
        case 71:     /* влево вверх */
          startx--;starty--;
          break;
        case 73:     /* вправо вверх */
          startx--;starty++;
```

```
        break;
      case 79:     /* влево вниз */
        startx++;starty--;
        break;
      case 81:     /* вправо вниз */
        startx++;starty++;
                        Глава II
        break;
      case 60:
           /* F2: отменить и вернуться к исходному размеру */
        startx = x;
        starty = y;
        ch = 59;
    }
    /* смотри при выходе за диапазон */
    if(startx<0) startx++;
    if(startx>=frame[num].endx) startx--;
    if(starty<0) starty++;
    if(starty>=frame[num].endy) starty--;
    deactivate(num);  /* стереть окно старого размера */
    frame[num].startx = startx;
    frame[num].starty = starty;
    window(num);      /* вывести окно с новым размером */
  } while(ch!=59);  /* F1 для подтверждения нового размера */
  deactivate(num);
}
/* Интерактивное перемещение окна */
void move(num)
int num;
{
  char ch;
  int x, y, ex, ey, startx, starty, endx, endy;
  /* активировать, если необходимо */
  if(!frame[num].active) window(num);
  startx = x = frame[num].startx;
  starty = y = frame[num].starty;
  endx = ex = frame[num].endx;
  endy = ey = frame[num].endy;
  window_xy(num, 0, 0);
  do {
    ch = get_special();
    switch(ch) {
      case 75:     /* влево */
        starty--;
        endy--;
        break;
      case 77:     /* вправо */
        starty++;
        endy++;
        break;
      case 72:     /* вверх */
        startx--;
        endx--;
        break;
      case 80:     /* вниз */
        startx++;
                        Глава II
        endx++;
        break;
      case 71:     /* влево вверх */
        startx--;starty--;
        endx--;endy--;
        break;
      case 73:     /* вправо вверх */
```

```
            startx--;starty++;
            endx--;endy++;
            break;
          case 79:     /* влево вниз */
            startx++;starty--;
            endx++;endy--;
            break;
          case 81:     /* вправо вниз */
            startx++;starty++;
            endx++;endy++;
            break;
  case 60:     /* F2: отменить и вернуться к исходному размеру */
            startx = x;
            starty = y;
            endx = ex;
            endy = ey;
            ch = 59;
        }
        /* смотри при выходе за диапазоном */
        if(startx<0) {
          startx++;
          endx++;
        }
        if(endx>=25) {
          startx--;
          endx--;
        }
        if(starty<0) {
          starty++;
          endy++;
        }
        if(endy>=79) {
          starty--;
          endx--;
        }
        /* стереть окно в старой позиции */
        deactivate(num);
        frame[num].startx = startx;
        frame[num].starty = starty;
        frame[num].endx = endx;
        frame[num].endy = endy;
        /* вывести окно в новую позицию */
        window(num);
        } while(ch!=59);  /* F1 для подьверждения изменения */
      deactivate(num);
    }
```

Глава II

При использовании как функции size(), так и функции move() после завершения изменения параметров окна нажмите клавишу F1. Окно будет иметь установленные размеры и положение при каждой последующей активации и до тех пор, пока вы не измените их снова. Для прерывания выполнения обеих функций используется клавиша F2, при этом окно сохраняет старые значения размеров и положения. При использовании этих функций допускается, чтобы окна, в момент изменения их размеров или положения необязательно были активными.

Глава II

Создание прикладных программ, использующих всплывающие окна

Работая с окнами очень важно помнить, что при вводе-выводе должны использоваться специальные оконные функции. Использование для этих целей стандартных функций Си чревато неприятностями, потому что создает возможность нарушения границы окна. Оконной функции, аналогичной по выполняемым действиям функции prinf(), разработано не было, и вы, возможно, захотите создать для этих

целей свою собственную функцию. Но простейший способ вывода в окно данных, тип которых отличен от символов и строк, заключается в использовании стандартной Си-функции sprintf() для преобразования любых типов данных в строку определенного формата, а затем в выводе этой строки в окно с помощью функции window_puts (). Аналогичный способ позволяет вводить данные, отличные от символов и строк. При этом функция window_gets() считывает данные, а приводит их к соответствующему типу стандартная Си-функция sscanf(), которая выполняет преобразование поступающих от клавиатуры данных.

Изображение в окне обычно формируется другой частью программы, не той, которая содержит функции управления окнами. Обычно это делается в функции main() или в функции инициализации, которая вызывается в начале программы. Предлагаем вашему вниманию три простые программы, использующие окна.

Глава II

Программа преобразования из десятичной в шестнадцатиричную систему счисления.

```
/* Десятично-шестнадцатиричный преобразоваель */
void dectohex()
{
  char in[80], out[80]
  int n;
  window(1);
  do {
    window_xy(1, 0, 0)  /* перейти к первой строке */
    window_cleol(1);    /* очистить строку */
    window_puts(1, "dec: "); /* промптер */
    window_gets(1, in);      /* считать число */
   window_putchar(1,  '\n'); /* перейти к следующей строке */
    window_cleol(1);          /* очистить ее */
 sscanf(in,"%d", &n); /* преобразовать во внутрений формат */
    sprintf(out, "%s%X", "hex: ",n);  /* преобразовать в
                          шестнадцатиричное представление */
   window_puts(1, out); /* вывести шестнадцатиричное число */
  } while(*in);
  deactivate(1);
}
```

Функция активирует свое окно, а затем в цикле принимает десятичные числа и выводит их шестнадцатиричные эквиваленты, до тех пор, пока пользователь не нажмет Ввод в ответ на запрос десятичного числа. Перед возвратом из функции ее окно деактивируется.

Глава II

Калькулятор с четырьмя функциями.

Очень подходящей и популярной областью использования всплывающих окон являются программы калькуляторов. Здесь представлена программа стекового калькулятора. Это означает, что при работе с ним вы должны сначала вводить операнды, а затем знак операции (т.н. постфиксная запись). Операнды помещаются в стек. В каждый момент времени выполняется операция над двумя операндами. При этом операнды извлекаются из стека, результат операции отображается и помещается в стек. Например, для того, вычислить результат выражения (10+5)/5, вы сначала должны ввести 10, затем 5, затем знак +. Результат этой операции, число 15, будет выведен на дисплей и помещен в вершину стека. Затем вы вводите 5 и знак /. Отображается результат 3. Стек рассчитан на 100 элементов. Можно вводить несколько операндов перед знаком операции. Функция calc(), а также подпрограммы push() и pop() для работы со стеком приводятся ниже. Хотя эта версия программы работает только с целыми числами, вы легко можете изменить ее таким образом, чтобы она работала с действительными числами.

```
    #define MAX 100
    int *p;  /* указатель стека */
    int *tos;  /* указатель вершины стека */
    int *bos;  /* указатель дна стека */
/* стековый, с постфиксной записью калькулятор с четырьмя
   функциями */
    void calc()
    {
      chra in[80], out[80];
      int answer, stack[MAX];
      int a,b;
      p = stack;
      tos = p;
      bos = p+MAX-1;
      window(2);
      do {
         window_xy(2, 0,0);
         window_cleol(2);
         window_puts(2, ": "); /* промптер калькулятора */
         window_gets(2, in);
         window_puts(2, "\n ");
         window_cleol(2);
         switch(*in) {
           case '+':
             a = pop();
             b = pop();
             answer = a+b;
             push(a+b);
             break;
           case '-':
             a = pop();
                              Глава II
             b = pop();
             answer = b-a;
             push(b-a);
             break;
           case '*':
             a = pop();
             b = pop();
             answer = b*a;
             push(b*a;
             break;
           case '/':
             a = pop();
             b=pop();
             if(a==0) {
                 window_putch("divide by 0\n");
                 break;
               }
               answer = b/a;
              break;
             default:
               push(atoi(in));
               continue;
             }
             sprintf(out, "%d", answer);
             window_puts(2, out);
           } while(*in);
           deactivate(2);
         }
       /* Поместить число в стек.  Возвратить 1 в случае успеха и
          0, если стек переполнен */
         push(i)
         int i;
```

```
        {
          if(p>bos) return 0;
          *p=i;
          p++;
          return 1;
        }
        /* Извлечь верхний элемент из стека.  Возвратить 0, если
           стек переполнен */
        pop()
        {
          p--;
          if(p<tos) {
            p++;
            return 0;
          }
          return *p;
       }
```


Глава II

Всплывающая записная книжка

Другой очень подходящей областью применения всплывающих окон являются программы типа "записная книжка". При использовании этих программ у вас может возникнуть потребность внести новую запись. Все, что от вас потребуется, это активизировать программу, внести новую запись и вернуться к тому, что вы делали ранее. Ниже представлен пример очень простой программы типа "записная книжка".

```
    #include "ctype.h"
    /* Всплывающая записная книжка */
    #define MAX_NOTE 10
    #define BKSP 8
    char notes[MAX NOTE][80];
    void notepad()
    {
      static firs=1;
      register int i, j;
      union inkey {
        char ch[2];
        int i;
      } c;
      char ch;
      /* Инициализировать массив записей, если это необходимо */
      if(frist) {
        for(i=0; i<MAX_note; i++)
          *note[i] = '\0';
        frist = !frist;
      }
      window(3);
      /* вывести на экран существующие записи */
      for(i=0; i<MAX_note; i++)  {
        if(*notes[i]) window_puts(3, notes[i]);
        window_putcar(3, '\n');
      }
      i=0;
      window_xy(3, 0, 0);
      for(;;) {
        c.i = window_getche(3);  /* считать символ, введенный с
                                                    клавиатуры */
        if(tolower(c.ch[1])==59 {  /* по F1 - завершение */
          deactivate(3);
          break;
        }
```

```
        /* если обычный символ, то внести его в запись */
        if(isprint(c.ch[0]) || c.ch[0]==BKSP) {
          window_cleol(3);
          notes[i][0] = c.ch[0];
          j = 1;
          window_putchar(3, notes[i][0]);
          do {
            ch = window_getche(3);
            if(ch==BKSP) {
              if(j>0) {
                j--;
                window_bksp(3);
              }
            }
            else {
              notes[i][j] = ch;
              j++;
            }
            } while(notes[i][j-1]!='\r');
            notes[i][j-1] = '\0';
            i++;
            window_putchar(3, '\n');
          }
          else {              /* это специальная клавиша */
            switch(c.ch[1]) {
              case 72:        /* стрелка вверх */
                if(i>0) {
                  i--;
                  window_upline(3);
                }
                break;
              case 80:        /* стрелка вниз */
                if(i<MAX_NOTE-1) {
                  i++;
                  window_dowline(3);
                }
                break;
              }
            }
          }
        }
```

Функция notepad() позволяет вводить до десяти строк. С помощью клавиш UP ARROW и DOWN ARROW вы можите перемещаться к нужной вам строке. Старое содержимое строки, в которую вносится новая запись, при этом стирается. Для выхода из программы "записная книжка" используется клавиша F1.

Глава II

Совместное использование всех программ

В этом разделе приводится программа, в которой используются все программы управления окнами и функции управления изображением, представленные в разделе 1, а также три прикладных программы, использующие окна. Программа имитирует работу редактора и позволяет вам с помощью функциональных клавиш активировать окна различного назначения или демонстрировать различные особенности работы с окнами. Вы можете немедленно ввести эту программу в вашу ЭВМ.

```
    /* Подпрограмма управления окнами и простая демонстрационная
    программа. Имитируется работа редактора. Три специальные
    оконные утилиты иллюстрируют мощь и очарование программ,
    использующих всплывающие окна. Этими утилитами являются:
    калькулятор с 4-мя функциями, десятично-шестнадцатиричный
    преобразователь и всплывающая записная книжка.    */
    #include "stdio.h"
```

```
#include "dos.h"
#include "stdlib.h"
#define BORDER 1
#define ESC 27
#define MAC_FRAME 10
#define REV_VID 0x70
#define NORM_VID 7
#define BKSP 8
void save_video(), restore_video(), pd_driver();
void goto_xy(, cls(), write_string(), write_char();
void display_header(), draw_border();
void window_gets(), size(), move(), window_cls();
void window_cleol(), window();
void dectohex(), notepad(), calc();
char far *vid_mem;
struct window_frame {
  int startx, endx, starty, endy;
  int curx, cury;
  unsigned char *p;
  char *header;
  iht border;
  int active;
} frame[MAX_FRAME];
main()
{
  union inkey {
  char ch[2];
  int i;
```



```
} c;
int i;
char ch;
cls();
goto_xy(0,0);
/* первым делом, создать рамки окна */
make_window(0, " Editor [Esc to exit] ", 0, 0, 24, 78, BORDER);
make_window(1, " Decimal to Hex ", 7, 40, 10, 70, BORDER);
make_window(2, " Calculator ", 8, 20, 12, 60, BORDER);
make_window(3, " Notepad [F1 to exit] ", 5, 20, 17, 60, BORDER);
/* использовать window() для активации описанного окна */
window(0);
do {
  c.i = window_getche(0);
  ch = c.i;      /* использовать только младший байт */
  if(ch=='\r')   /* должен выполнять переход к началу
                                    следующей строки */
      window_putchar(0, '\n');
  switch(c.ch[1]) {   /* см. при использовании стрелок или
                         функциональных клавиш */
    case 59: /* F1 демонстрирует работу функции window() */
      window(1);
      for(i=0; i<10; i++)
        if(window_xy(1, i, i)) window_putchar(1,'X');
        getch();
        deactivate(1);
        break;
        case 60: /* F2 демонстрирует изменение размера и
                                          положения окна */
        size(1);
        move(1);
        break;
      case 61:  /* F3 вызывает калькулятор */
        calc();
        break;
```

```
      case 62:    /* F4 вызывает десятично-шестнадцатиричный
                     преобразователь */
        dectohex();
        break;
      case 63:  /*  F5 вызывает записную книжку */
        notepad();
        break;
        case 72:  /* вверх */
        window_upline(0);
        break;
      case 80:    /* вниз */
        window_downline(0);
        break;
      }
```



```
    } while (ch!=ESC);
    deactivate(0);  /* удалить окно */
  }
/*********************************************************/
/*  Оконные функции                                      */
/*********************************************************/
/* Вывести на экран спускающееся окно */
void window(num)
int num;  /* номер окна */
{
  int vmode, choice;
  int x, y;
  vmode = video_mode();
  if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
    printf("video must be in 80 column text mode");
    exit(1);
  }
  /* установить соответствующий адрес видеопамяти */
  if(vmode==7) vid_mem = (char far *) 0xb0000000;
  else vid_mem = (char far *) 0xb0000000;
  if(!frame[num].active) {  /* используется непостоянно */
    save_video(num);/* сохранить текущее содержимое экрана */
    frame[num].active = 1;  /* установить флаг активности */
  }
  if(frame[num].border) draw_border(num);
  dispay_header(num);     /* вывести окно на экран */
  x = frame[num].startx + frame[num].curx + 1;
  y = frame[num].starty + frame[num].cury + 1;
  goto_xy(x, y);
}
/* Создать рамку спускающегося окна. Возвратить 1, если рамка
окна может быть создана и 0 в противном случае */
make_window(num, header, startx, starty, endx, endy, border)
int num;         /* номер окна */
char *header;    /* текст заголовка */
int startx, starty; /* координаты x,y верхнего левого угла */
int endx, endy;     /* координаты x,y нижнего правого угла */
int border;           /* без бордюра, если 0 */
{
  unsigned char *p;
  if(num>MAX_FRAME) {
    printf("Too many windows\n");
```



```
    return 0;
  }
if((startx>24) || (startx<0) || (starty>78) || (starty<0)) {
    printf("range error");
    return 0:
  }
```

```
      if((endx>24) || (endy>79)) {
        printf("window won't fit");
        return 0;
      }
      /* отвести достаточное количество памяти */
 p = (unsigned char *) malloc(2*(endx-startx+1)*(endy-starty=1));
if(!p) exit(1);/* используйте ваш собственный обработчик ошибок*/
      /* создать рамку */
      frame[num].startx = startx; frame[num].endx = endx;
      frame[num].starty = starty; frame[num].endy = endy;
      frame[num].p = p;
      frame[num].header = header;
      frame[num].border = border;
      frame[num].active = 0;
      frame[num].curx = 0; frame[num].cury = 0;
      return 1;
    }
    /* редактировать окно и удалить его с экрана */
    deactivate(num)
    int num;
    {
      /* установить курсор в левый верхний угол */
      frame[num].curx = 0;
      frame[num].cury = 0;
      restore_video(num);
    }
    /* Интерактивное изменение размеров окна */
    void size(num)
    int num;
    {
      char ch;
      int x, y, startx, starty;
      /* активировать, если необходимо */
      if(!frame[num].active) window(num);
      startx = x = frame[num].startx;
      starty = y = frame[num].starty;
      window_xy(num, 0, 0);
      do {
                              Глава II
        ch = get_special();
        switch(ch) {
          case 75:    /* влево */
            starty--;
            break;
          case 77:    /* вправо */
            starty++;
            break;
          case 72:    /* вверх */
            startx--;
            break;
          case 80:    /* вниз */
            startx++;
            break;
          case 71:    /* влево вверх */
            startx--; starty--;
            break;
          case 73:    /* вправо вверх */
            startx--; starty++;
            break;
          case 79:    /* влево вниз */
            startx++; starty--;
            break;
          case 81:    /* вправо вниз */
            startx++; starty++;
```

```
        break;
      case 60:    /* F2: отменить и вернуться к исходным
                                                  размерам */
        startx = x;
        starty = y;
        ch = 59;
    }
    if(startx<0) startx++;
    if(startx>=frame[num].endx) startx--;
    if(starty<0) starty++;
    if(starty>=frame[num].endy) starty--;
    deactivate(num);
    frame[num].startx = startx;
    frame[num].starty = starty;
    window(num);
  } while(ch!=59);
  deactivate(num);
}
/* Интерактивное перемещение окна */
void move(num)
int num;
{
  char ch;
  int x, y, ex, ey, startx, starty, endx, endy;
  /* активировать, при необходимости */
                        Глава II
  if(!frame[num].active) window(num);
  startx = x = frame[num].startx;
  starty = y = frame[num].starty;
  endx = ex = frame[num].endx;
  endy = ey = frame[num].endy;
  window_xy(num, 0, 0);
  do {
    ch = get_special();
    switch(ch) {
      case 75:   /* влево */
      starty--;
      endy--;
      break;
      case 77:   /* вправо */
      starty++;
      endy++;
      break;
      case 72:   /* вверх */
      startx--;
      endx--;
      break;
      case 80:   /* вниз */
      startx++;
      endx++;
      break;
      case 71:   /* влево вверх */
      startx--; starty--;
      endx--; endy--;
      break;
      case 73:   /* вправо вверх */
      startx--; starty++;
      endx--; endy++;
      break;
      case 79:   /* влево вниз */
      startx++; starty--;
      endx++; endy--;
      break;
      case 81:   /* вправо вниз */
```

```
        startx++; starty++;
        endx++; endy++;
        break;
        case 60:    /* F2: отменить и вернуться к исходным
                                                    размерам */
          startx = x;
          starty = y;
          endx = ex;
          endy =ey;
          ch = 59;
  }
  /* см. при выходе за диапазон */
  if(startx<0) {
                          Глава II
    startx++;
    endx++;
  }
  if(endx>=25) {
    startx--;
    endx--;
    }
    if(starty<0) {
      starty++;
      endy++;
    }
    if(endy>=79) {
      starty--;
      endy--;
    }
    deactivate(num);
    frame[num].startx = startx;
    frame[num].starty = starty;
    frame[num].endx = endx;
    frame[num].endy = endy;
    window(num);
  } while(ch!=59);
  deactivate(num);
}
  /* Вывести текст заголовка, начиная с определенной позиции */
  void display_header(num)
  int num;
  {
    register int y, len;
    y = frame[num].starty;
    /* Вычислить начальную позицию относительно центра текста
    заголовка, если отрицательная, то текст не подходит */
    len = strlen(frame[num].header);
    len = (frame[num].endy - y - len) / 2;
    if(len<0) return;  /* не выводить на экран */
    y = y + len;
    write_string(frame[num].startx, y,
                frame[num].header, NORM_VID);
  }
  void draw_border(num)
  int num;
  {
    register int i;
    char far *v, far *t;
    v = vid_mem;
    t = v;
    for(i=frame[num].startx+1; i<frame[num].endx; i++) {
                          Глава II
    v += (i*160) + frame[num].starty*2;
    *v++ = 179;
```

```
      *v = NORM_VID;
      v = t;
      v += (i*160) + frame[num].endy*2;
      *v++ = 179;
      *v = NORM_VID;
      v = t;
    }
    for(i=frame[num].starty+1; i<frame[num].endy; i++) {
       v += (frame[num].startx*160) + i*2;
       *v++ = 196;
       *v = NORM_VID;
       v = t;
       v+ =(frame[num].endx*160) + i*2;
       *v++ = 196;
       *v = NORM_VID;
       v = t;
    }
 write_char(frame[num].startx, frame[num].starty, 218, NORM_VID);
 write_char(frame[num].startx, frame[num].endy, 191, NORM_VID);
 write_char(frame[num].endx, frame[num].starty, 192, NORM_VID);
    write_char(frame[num].endx, frame[num].endy, 217, NORM_VID);
    }
/*************************************************************/
/*   Оконные функции ввода/вывода                            */
/*************************************************************/
/* Вывести символ в текущую позицию курсора в созданном окне.
   Возвратить 0, если окно не активное и 1 в противном случае. */
    window_puts(num, str)
    int num;
    char *str;
    {
      /* убедиться, что окно активное */
      if(!frame[num].activite) return 0;
      for( ; *str; str++)
        window_putchar(num, *str);
      return 1;
    }
 /* Вывести символ в текущую позицию курсора в созданном окне.
    Возвратить 0, если окно не активное и 1 в противном случае */
    window_putchar(num, ch)
    int num;
    char ch;
    {
      register int x, y;
      char far *v;
                              Глава II

      /* убедиться, что окно активное */
      if(!frame[num].active) return 0;
      x = frame[num].curx + frame[num].startx + 1;
      y = frame[num].cury + frame[num].starty + 1;
      v = vid_mem;
      v += (x*160) + y*2;  /* вычислить адрес */
      if(y>=frame[num].endy) {
        return 1;
      }
      if(x>=frame[num].endx) {
        return 1;
      }
      if(ch=='\n') {   /* символ перехода на следующую строку */
        x++;
        y = frame[num].startx+1;
        v = vid_mem;
        v += (x*160) + y*2;  /* вычислить адрес */
```

```
    frame[num].curx++;   /* нарастить x */
    frame[num].cury = 0; /* спросить y */
  }
  else {
    frame[num].cury++;
    *v++ = ch;  /* вывести символ */
    *v++ = NORM_VID; /* нормальные видеоатрибуты */
  }
  window_xy(num, frame[num].curx, frame[num].cury);
  return 1;
}
/* Установить курсор в оределенной позиции окна.
   Возвратить 0 при выходе за границу, не ноль в противном
   случае */
window_xy(num, x, y)
int num, x, y;
{
  if(x<0 || x+frame[num].startx>=frame[num].endx-1)
    return 0;
  if(y<0 || y+frame[num].starty>=frame[num].endy-1)
    return 0;
  frame[num].curx = x;
  frame[num].cury = y;
  goto_xy(frame[num].startx+x+1, frame[num].starty+y+1);
  return 1;
}
/* Считать строку из окна */
void window_gets(num, s)
int num;
char *s;
{
```



```
  char ch, *temp;
  temp = s;
  for(,,) {
    ch = window_getche(num);
    switch(ch) {
      case '\r':  /* нажата клавиша ENTER */
        *s='\0';
        return;
      case BKSP:  /* возврат */
        if(s>temp) {
          s--;
          frame[num].cury--;
          if(frame[num].cury<0) frame[num].cury = 0;
            window_xy(num, frame[num].curx, frame[num].cury);
             write_char(frame[num].startx+ frame[num].curx+1;
         frame[num].starty+frame[num].cury+1, ' ', NORM_VID);
        }
        break;
      default: *s = ch;
        s++;
    }
  }
}
/* Ввести символ в окно с клавиатуры.
   Возвратить полный 16-ти разрядный скан-код */
window_getche(num)
int num;
{
   union inkey {
     char ch[2];
     int i;
```

```
    } c;
    if(!frame[num].active) return 0;  /* окно не активное */
    window_xy(num, frame[num].curx, frame[num].cury);
    c.i = bioskey(0);   /* ввести символ с клавиатуры */
    if(c.ch[0]) {
      switch(c.ch[0]) {
        case '\r':    /* нажата клавиша ENTER */
          break;
        case BKSP:   /*возврат */
          break;
        default:
  if(frame[num].cury+frame[num].starty < frame[num].endy-1) {
  write_char(frame[num].startx+ frame[num].curx+1,
  frame[num].starty+frame[num].cury+1, c.ch[0], NORM_VID);
           frame[num].cury++;
                       Глава II

         }
      }
   if(frame[num].curx < 0) frame[num].curx = 0;
   if(frame[num].curx+frame[num].startx > frame[num].endx-2)
        frame[num].curx--;
      window_xy(num, frame[num].curx, frame[num].cury);
    }
    return c.i;
  }
  /* Очистить окно */
  void window_cls(num)
  int num;
  {
    register int i,j;
    char far *v, far *t;
    v = vid_mem;
    t = v;
    for(i=frame[num].starty+1; i<frame[num].endy; i++)
      for(j=frame[num].startx+1; j<frame[num].endy; j++) {
        v = t;
        v += (j*160) + i*2;
        *v++ = ' ';    /* вывести пробел */
        *v = NORM_VID; /* нормальные видеоатрибуты */
    }
    frame[num].curx = 0;
    frame[num].cury = 0;
  }
  /* Очистить до конца строки */
  void window_cleol(num)
  int (num);
  {
  register int i, x, y;
  x = frame[num].curx;
  y = frame[num].cury;
  window_xy(num, frame[num].curx, frame[num].cury);
  for(i=frame[num].cury; i<frame[num].endy-1; i++)
    window_putchar(num,' ');
  window_xy(num, x, y);
}
/* Переместить курсор на одну строку вверх.
   Возвратить ненулевой код в случае успеха, 0 - в пртивном
   случае. */
window_upline(num)
int num;
{
  if(frame[num].curx>0) {
    frame[num].curx--;
```

```
        window_xy(num, frame[num].curx, frame[num].cury);
        return 1;
      }
      return 0;
    }
    /* Переместить курсор на одну строку вниз.
       Возвратить ненулевой код в случае успеха, 0 - в противном
       случае. */
    window_dowline(num)
    int num,
    {
      if(frame[num].curx<frame[num].endx-frame[num].startx-1) {
        frame[num].curx++;
        window_xy(num, frame[num].curx, frame[num].cury);
        return 1;
      }
      return 1;
      /* стереть предыдущий символ
      window_bksp(num)
      int (num);
      {
      if(frame[num].cury>0) {
        frame[num].cury--;
        window_xy(num, frame[num].curx, frame[num].cury);
        window_putchar(num, ' ');
        frame[num].cury--;
        window_xy(num, frame[num].curx, frame[num].cury);
      }
    }
/*****************************************************************/
/*   Дополнительные функции                                      */
/*****************************************************************/
    /* Вывести на экран строку с дополнительными атрибутами */
    void write_strihg(x, y, attrib)
    int x, y;
    char *p;
    int attrib;
    {
      register int i;
      char far *v;
      v = vid_mem;
      v += (x*160) + y*2; /* вычислить адрес */
      for(i=y; i++) {
        *v++ = *p++;  /* вывести символ */
        *v++ = attrib; /* вывести атрибуты */
      }
```



```
    }
    /* Вывести символы с определенными атрибутами */
    void write_char(x, y, ch, attrib)
    int x, y;
    char ch;
    int attrib;
    {
      register int i;
      char far *v;
      v = vid_mem;
      v += (x*160) + y*2;
      *v++ = ch;   /* вывести символ */
      *v = attrib; /* вывести атрибуты */
    }
```

```
    /* Сохранить содержимое части экрана */
    void save_video(num)
    int num;
    {
      register int i, j;
      char *buf_ptr;
      char far *v, far *t;
      buf_ptr = frame[num].p;
      v = vid_mem;
      for(i=frame[num].starty; i<frame[num].endy+1; i++)
        for(j=frame[num].startx; j<frame[num].endx+1; j++) {
          t = (v + (j*160) + i*2);
          *buf_ptr++ = *t++;
          *buf_ptr++ = *t;
          *(t-1) = ' ';  /* очистить окно */
        }
    }
    /* Восстановить содержимое части экрана */
    void restore_video(num)
    int num;
    {
      register int i,j;
      char far *v, far *t;
      char *buf_ptr;
      buf_ptr = frame[num].p;
      v = vid_mem;
      t = v;
      for(i=frame[num].starty; i<frame[num].endy+1; i++)
        for(j=frame[num].startx; i<frame[num].endx+1; j++) {
          v = t;
          v += (j*160) + i*2;
          *v++ = buf_ptr++;   /* вывести символ */
          *v = *buf_ptr++;    /* вывести атрибуты */
                              Глава II

      }
      frame[num].active = 0;
    }
    /* Очистить экран */
    void cls()
    {
      union REGS r;
      r.h.ah=6;   /* код прокрутки экрана */
      r.h.al=0;   /* код очистки экрана */
      r.h.ch=0;   /* начальный ряд */
      r.h.cl=0;   /* начальный столбец */
      r.h.dh=24;  /* конечный ряд */
      r.h.dl=79;  /* конечный столбец */
      r.h.bh=7;   /* пустая строка - черная */
      int86(0x10, &r, &r);
    }
    /* Установить курсор в позицию с координатами x, y */
    void goto_xy(x,y)
    int x,y;
    {
      union REGS r;
      r.h.ah=2; /* функция адресации курсора */
      r.h.dl=y; /* координаты столбца */
      r.h.dh=x; /* координаты ряда */
      r.h.bh=0; /* страница видеопамяти */
      int86(0x10, &r, &r);
    }
/* Возвратить позиционный код стрелки и функциональных клавиш */
    get_special()
```

```
    {
      union inkey {
        char ch[2];
        int i;
      } c;
      /* while(!bioskey(1)) ; /* ждать нажатия клавиши */
      c.i = bioskey(0);       /* считать код нажатой клавиши */
      return c.ch[1];
    }
    /* Возвратить код текущего видеорежима */
    video_mode()
    {
      union REGS r;
      r.h.ah = 15;  /* получить код видеорежима */
      return int86(0x10, &r, &r) & 255;
    }
```



```
    is_in(s, c)
    char *s, c;
    {
      register int i;
      for(i=0; *s; i++) if(*s++==c) return i+1;
      return 0;
    }
    #include "ctype.h"
/*****************************************************************/
/*   Функции управления всплывающими окнами                      */
/*****************************************************************/
    #define MAX 100
    int *p;  /* указатель стека */
    int *tos; /* указатель вершины стека */
    int *bos; /* указатель дна стека */
/* Стековый, с постфиксной записью калькулятор с 4-мя функциями*/
    void calc()
    {
      char in[80], out[80];
      int answer, stack[MAX];
      int a,b;
      p = stack;
      tos = p;
      bos = P+MAX-1;
      window(2);
      do {
        window_xy(2, 0, 0);
        window_cleol(2);
        window_puts(2, ": "); /* промтер калькулятора */
        window_gets(2, in);
        window_puts(2, "\n");
        window_cleol(2);
        switch(*in) {
          case '+':
            a = pop();
            b = pop();
            answer = a+b;
            push(a+b);
            break;
          case '-':
            a = pop();
            b = pop();
            answer = b-a;
            push(b-a);
```

```
            break;
          case '*':
            a = pop();
            b = pop();
            answer = b*a;
            push(b*a);
            break;
            case '/':
              a = pop();
              b=pop();
              if(a==0) {
                  window_puts("divide by 0\n");
                  break;
              }
              ansver = b/a;
              push(b/a);
              break;
            default:
              push(atoi(in));
              continue;
          }
          sprintf(out, "%d", answer);
          window_puts(2, out);
        } wile(*in);
        deactivate(2);
      }
      /* Поместить число в стек.
         Возвратить 1 в случае успеха и 0 в противном случае */
      push(i)
      int i;
      {
        if(p>bos) return 0;
        *p=i;
        p++;
        return 1;
      }
      /* Извлечь верхний элемент из стека.
         Возвратить 0 если стек пуст */
      pop()
      {
        p--;
        if(p<tos) {
          p++;
          return 0;
        }
        return *p;
      }
```

Глава II

```
      /* Десятично-шестнадцатиричный преобразователь */
      void dectohex()
      {
        char in[80], out[80];
        int n;
        window(1);
        do {
window_xy(1, 0, 0);  /* перейти к первой строке */
window_cleol(1);     /* очистить строку */
window_puts(1, "dec: ");  /* промтер */
window_gets(1, in);       /* считать число */
window_putchar(1, '\n');  /* перейти к следующей строке */
window_cleol(1);          /* очистить ее */
sscanf(in,"%d", &n);      /* преобразовать во внутренний формат*/
sprintf(out, "%s%X", "hex: ",n); /* преобразовать в
```

```
                              шестнадцатиричное представление */
  window_puts(1, out); /* вывести шестнадцатиричное число */
   } wile(*in);
   deactivate(1);
 }
 /* Всплывающая записная книжка */
 #define MAX_NOTE 10
 #define BKSP 8
 char notes[MAX_NOTE][80];
 void notepad()
 {
   static first=1;
   register int i; j;
   union inkey {
     char ch[2];
     int i;
   } c;
   char ch;
/* инициализировать массив записей, если это необходимо */
   if(frist) {
     for(i=0; i<MAX_NOTE; i++)
       *notes[i] = '\0';
     first = !first;
 }
 window(3);
 /* вывести на экран существующие записи */
 for(i=0; i<MAX_NOTE; i++) {
   if(*notes[i]) window_puts(3, notes[i]);
   window_putchar(3, '\n');
 }
 i=0;
```



```
 window_xy(3, 0, 0);
 for(;;) {
   c.i = bioskey(o);  /* считать код клавиши */
   if(tolower(c.ch[1])==59) {  /* F1 - для входа */
     deactivate(3);
     break;
   }
   /* если обычная клавиша */
   if(isprint(c.ch[0]) || c.ch[0]==BKSP) {
     window_cleol(3);
     notes[i][0] = c.ch[0];
     j = 1;
     window_putchar(3, notes[i][0]);
     do {
       ch = window_getche(3);
       if(ch==BKSP) {
         if(j>0) {
           j--;
           window_bksp(3);
             }
           }
           else {
             notes[1][j] = ch;
             j++;
           }
         } while(notes[i][j-1]!='\r');
         notes[i][j-1] = '\0';
         i++;
         window_putchar(3, '\n');
       }
       else {    /* если специальная клавиша */
```

```
            switch(c.ch[1]) {
              case 72:     /* стрелка вверх */
                if(i>0) {
                  i--;
                  window_upline(3);
                }break
              case 80:     /* стрелка вниз */
                if(i<MAX_NOTE-1) {
                  i++;
                  window_dowline(3);
                }
                break;
            }
          }
        }
      }
```

Пять первых функциональных клавиш выполняют следующие действия:

F1 - демонстрирует работу функции window_xy()
F2 - демонстрирует изменение размера и положения окна
F3 - вызывает калькулятор
F4 - вызывает программу десятично-шестнадцатиричного преобразования
F5 - вызывает "записную книжку".

Рисунки 2-1, 2-2 и 2-3 иллюстрируют появление на экране соответствующих меню. Рис. 2-4 иллюстрирует изменение размеров и положения окна программы десятично-шестнадцатиричного преобразования.

Модификации программ управления окнами

При использовании представленных в данном разделе программ управления окнами вы всегда должны точно определять окно, с которым собираетесь работать. Этот способ при использовании программ обеспечивает большую гибкость. Другой способ управления окнами основан на применении стека.

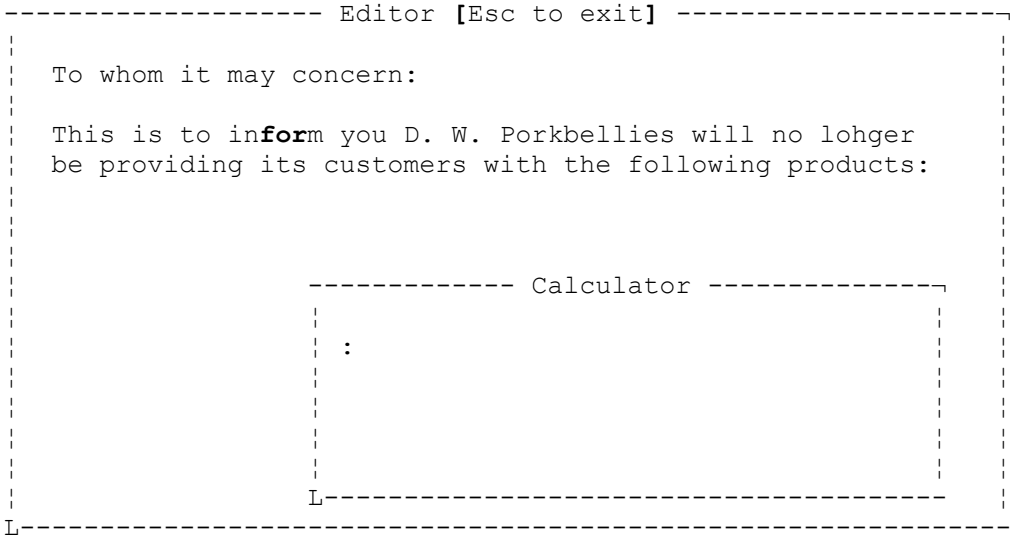
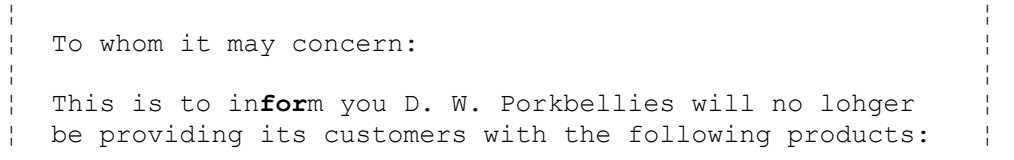


Рис. 2-1 Окно калькулятора.



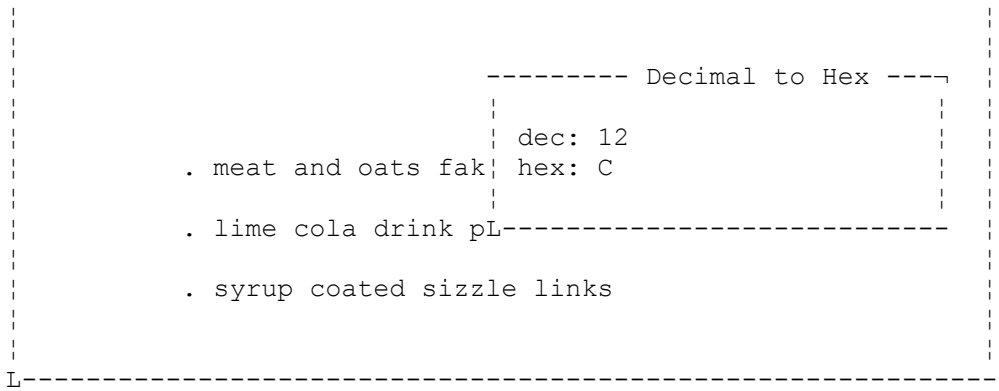


Рис. 2-2 Окно десятично-шестнадцатиричного преобразователя.

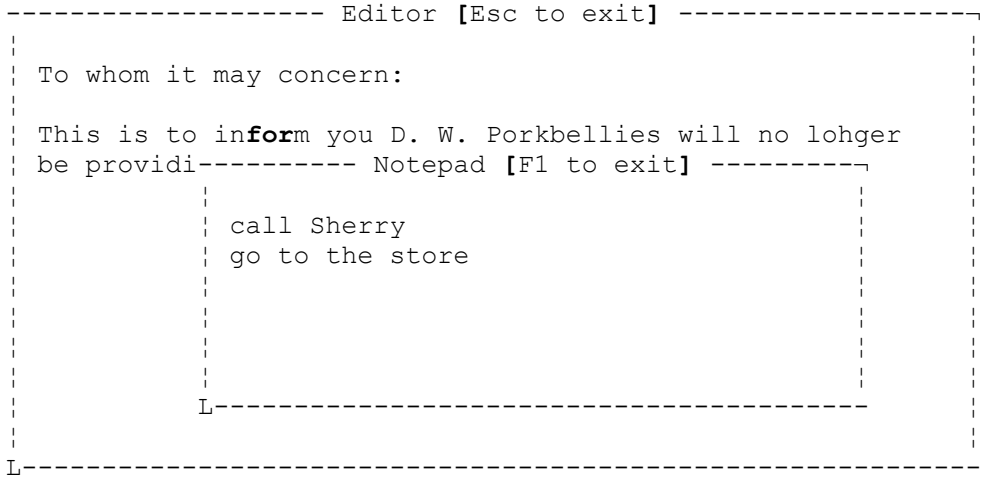


Рис. 2-3 Окно записной книжки.

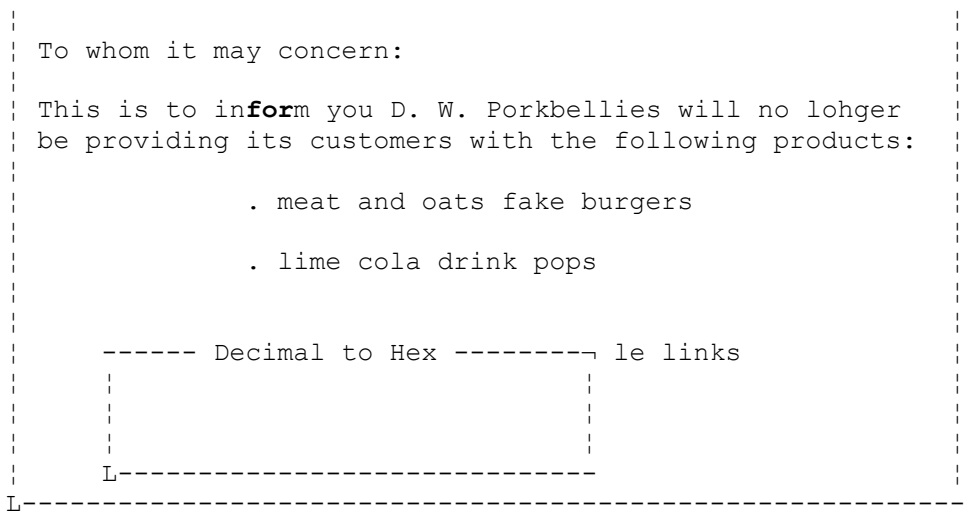


Рис. 2-4 Изменение размеров и положения окна десятично-шестнадцатиричного преобразователя.

В этом случае оконные программы ввода-вывода не получают номер окна в качестве аргумента. Вместо этого номера окон помещаются в стек в том порядке, в котором они были активированы. Оконные программы всегда работают с тем окном, номер которого находится в вершине стека. При деактивации окна его номер извлекается из стека. Преимущество этого способа заключается в том, что вы не должны в этом случае думать о номерах окон. Вы можете модифицировать программы управления окнами, чтобы они

работали именно этим способом.

Другая модификация может заключаться в том, чтобы обеспечивать прокрутку окна, когда курсор достигает его нижней границы. В существующем варианте, если курсор достиг нижней границы, то после нажатия клавиши Ввод ничего не происходит. Вы однако можете изменить программы таким образом, чтобы верхняя строка пропадала, а внизу появлялась новая пустая строка.

Наконец, те читатели, которые имеют цветные дисплеи, могут использовать различные цвета для обозначения границ различных окон. При правильном применении это добавит привлекательности вашим программам.

ГЛАВА 3

Программы, остающиеся резидентными после завершения и формирующие при их вызове всплывающие изображения на экране дисплея.

Простая на первый взгляд идея создания программ, которые оставались бы резидентными в памяти после их завершения и реагировали на вызов формированием всплывающих изображений на экране дисплея, на самом деле является одной из наиболее трудных задач программирования для ПЭВМ. Такие программы называются TSR-программами. При чтении данного раздела вы должны получше пристегнуться ремнями безопасности и одеть защитный шлем, т.к. создание TSR-программ связано с риском. Но этот риск оправдан возможным вознаграждением - поистине профессиональными результатами, которыми гордился бы любой программист мирового класса.

Поскольку TSR-программы, естественно, должны на низком уровне взаимодействовать с аппаратурой и операционной системой, то излагаемые в данном разделе сведения будут применимы только к ПЭВМ линии IBM PC, работающими под операционной системой DOS. По причинам, которые будут указаны ниже, приводимые в разделе программы рассчитаны на компилятор Turbo C, но могут быть модифицированы и для других компиляторов.

Предупреждение. Для разработки и использования TSR-программ характерна модификация таблицы векторов прерываний. Приведенные в данном разделе программы транслируются с помощью Турбо Си версии 1.0 и в этом случае работают корректно и без посторонних эффектов. При использовании другого компилятора корректность работы не гарантируется. Кроме того, если вы будете набирать эти программы вручную, то можете внести свои ошибки. И в том, и в другом случае это может привести к фатальному сбою системы, в результате чего могут быть уничтожены данные на вашем винчестерском диске. Поэтому целесообразно делать резервные копии файлов. Я уверен, что приводил к краху мою модель 60 не менее 100 раз за те два дня, пока отлаживал основную логику своей программы. **(**К счастью, я не затирал при этом винчестерского диска**)**.

Что такое TSR-программа?

TSR-программы создаются путем вызова функции 49 DOS, по которой производится возврат из программы в DOS. При этом программа остается в области памяти, которую DOS в дальнейшем не использует. Таким образом, программа может быть мгновенно вызвана без повторной загрузки. Одним из многих широко известных примеров TSR-программ является программа Sidekick фирмы Borland.

Большинство TSR-программ вызываются с помощью прерывания, которое может быть сформировано несколькими способами. Наиболее распространенными являются прерывания по таймеру, прерывания клавиатуры и печати экрана. Для TSR-программ, формирующих изображение на экране, обычно используются прерывания от клавиатуры или печати экрана, поскольку позволяют пользователю вызывать TSR-программу путем одиночного нажатия клавиши.

Прерывания в семействе процессоров 8086.

Процессоры семейства 8086 поддерживают до 256 различных прерываний по вектору. Прерывание по вектору вызывает выполнение программы обработки прерываний (ISR), адрес которой содержится в таблице векторов прерываний. Хотя некоторые старшие процессоры семейства требуют, чтобы программы обработки прерывания располагались в определенных адресах памяти, механизм прерываний по вектору позволяет определять адреса программ обработки прерываний.

Таблица векторов начинается с адреса 0000:0000 и ее размер составляет 1024 байта. Поскольку адрес программы обработки прерывания может быть любым, то для его определения требуется 32 разряда (4 байта). Следовательно, размер каждой записи в таблице векторов составляет 4 байта. Адреса ISR-программ в таблице записываются таким образом, что адрес программы обработки прерывания 0 находится по адресу 0000:0000, программы обработки прерывания прерывания 1 - по адресу 0000:0004, прерывания 2 - по адресу 0000:0008 и т.д.

Когда происходит прерывание, то любые другие прерывания запрещаются. Ваша программа обработки прерывания сразу после того, как она начнет выполняться, должна разрешить прерывания, чтобы избежать краха системы. Программа обработки прерывания должна завершаться командой IRET.

Прерывания против DOS и BIOS: Тревога в стране DOS.

Программисты часто выражают недовольство тем, что DOS не является повторно входимой программой. Это означает, что когда одна программа обращается к DOS, то другая программа этого делать не может. (Этим объясняется, в частности, почему DOS не является мультизадачной операционной системой). Таким образом, программа обработки прерывания не может вызывать никакой функции DOS, такая попытка приводит к краху системы. Поэтому программа обработки прерывания должна сама выполнять те действия, которые производятся при обращении к функциям DOS. К счастью, для формирования видеоизображения мы можем использовать программы непосредственного обращения к видеопамяти из разделов 1 и 2.

BIOS допускает некоторую повторную входимость. Например, прерывание 16, соответствующее вводу с клавиатуры, может быть использовано в этом режиме без каких-либо побочных эффектов. Некоторые другие подпрограммы использовать таким образом не столь безопасно. Обнаружить это можно только экспериментальным путем. О том, что функцию нельзя использовать в таком режиме, вы узнаете по фатальному сбою системы. Для приведенных в данном разделе примеров и для многих распространенных в мире программ прерывания 16 вполне достаточно.

Поскольку многие из функций стандартной библиотеки языка Си обращаются к DOS или к BIOS, то они не должны использовать тех функций DOS и BIOS, которые не обеспечивают повторной входимости. Следует помнить, что не только функции ввода-вывода обращаются к DOS и BIOS. Например, функция распределения памяти malloc() обращается к DOS для определения размера свободной памяти в системе. К сожалению, программы, которые рассчитаны на один компилятор, могут не работать с другим компилятором. Этим и объясняется, почему TSR-программы так трудно создавать и переносить в другую среду и почему TSR-программ создано столь немного при их очень большой популярности.

По существу, вы должны воспринимать TSR-программы как "заблудшие" программы, о существовании которых DOS не подозревает. И в дальнейшем, чтобы сохранить тайну о своем существовании эти программы должны избегать любого взаимодействия

с DOS. Всего пары обращений к DOS достаточно, и вашей программе будет устроена кровавая резня. Чтобы этого избежать, вы должны ощущать себя шпионом и иметь нервы автогонщика.

Модификатор функций прерывания Турбо Си.

Хотя стандарт ANSI этого и не требует, Турбо Си включает специальный модификатор типа функции, который называется **int**errupt и позволяет использовать функции Си в качестве TSR-программ. (Большинство основных разработчиков компиляторов Си по всей вероятности включат это средство в свои будущие разработки, поскольку это очень важное расширение). Например, предположим, что функция test() используется для обработки прерываний. В этом случае вы должны определить ее так, как показано ниже. Параметры, описывающие значения соответствующих регистров во время прерывания, не нужно определять, если они не будут использоваться.

```
void interrupt test(bp, di, si, ds, es, dx, cx, bx,
                    ax, ip, cs, flags)
unsigned bp, di, si, ds, es, dx, cx, bx, ax, ip, cs, flags;
{
    .
    .
    .
}
```

Функция **int**errupt автоматически сохраняет значения всех регистров и восстанавливает их перед возвратом управления вызывающей программе. Эта функция использует для возврата управления команду IRET вместо обычной в таком случае команды RET.

В представленных в данной книге примерах модификатор **int**errupt применяется только для тех функций, которые используются в качестве точек входа в программы обработки прерываний TSR-программ.

Если ваш компилятор не поддерживает модификатор **int**errupt, то вам необходимо написать на ассемблере небольшой интерфейсный модуль, который будет сохранять значения регистров, переустанавливать разрешение прерываний, а затем вызывать соответствующую функцию Си. Для выхода из модуля необходимо использовать команду IRET. Средства создания функций на языке ассемблера различны для разных компиляторов, так что читайте имеющееся у вас руководство пользователя.

Общий план TSR-программы

Все TSR-программы обычно состоят из двух разделов. Первая часть используется для инициализации TSR-программы и возврата управления DOS путем использования реентерабельного системного вызова. Эта часть не выполняется до тех пор, пока не возникает необходимость в перезагрузке программы. При этом производится запись адреса точки входа TSR-программы в соответствующее место таблицы векторов.

Вторая, прикладная часть, занимается формированием изображений. При этом почти всегда используются окна, а следовательно,и программы управления окнами. При этом изображение на экране восстанавливается после завершения работы прикладной части программы. Следует помнить, что у большинства TSR-программ прикладные части представляют собой утилиты формиривания изображения, как у программы типа "записной книжки" или "калькулятора". После своего завершения они восстанавливают изображение на экране в том же виде, каким оно было перед запуском этих программ.

Использование прерывания печати экрана.

Без сомнений, прерыванием, которое наиболее просто "украсть" у DOS, является прерывание номер 5. Это прерывание вызывается при нажатии клавиши PT SCR. Если вы готовы пожертвовать функцией печати экрана, то можете заменить адрес этой программы в таблице векторов адресом вашей TSR-программы. Таким образом, при каждом нажатии клавиши PT SCR будет вызываться ваша TSR-программа.

Примером такой программы является резидентный калькулятор. Программы для работы с окнами и программа калькулятора из раздела 2 приводятся здесь с некоторыми небольшими изменениями.

Раздел инициализации

Раздел инициализации программы резидентного калькулятора очень небольшой и целиком помещается в нижеследующей функции main().

```
void interrupt tsr_ap(); /* вход в прикладную программу */
main()
{
  struct address {
    char far *p;
  } ;
  /* адрес прерывания печати экрана */
  struct address far *addr = (struct address far *) 20;
  addr->p = (char far *) tsr_ap;
  set_vid_mem();
  tsr(2000);
}
```

TSR-программа первым делом должна заменить адрес программы обработки прерывания 5 указателем функции, определенной в самой TSR-программе. Есть несколько способов изменения адреса в таблице векторных прерываний. Один из способов состоит в использовании системного вызова DOS. Однако неудобство использования функции DOS заключается в том, что она требует задания значения адресного сегмента в регистре ES, который недоступен при использовании функции int86(). Некоторые компиляторы, как например Турбо Си, включают специальные функции, предназначенные для установки адреса в таблице прерываний. Однако способ, предлагаемый здесь, будет работать при использовании практически любого компилятора. Функция tsr_ap() является точкой входа в прикладную часть TSR-программы. Она использует указатель на содержимое таблицы векторов, соответствующее прерыванию 5. (Напоминаем, что вектор 5 расположен по адресу 20(4x5) в таблице, поскольку каждый вектор имеет размер 4 байта. Некоторые TSR-программы восстанавливают исходное значение адреса. Но при использовании приводимых здесь программ вы должны будете перезагружать систему, чтобы восстановить исходные значения векторов прерываний.

В предыдущих разделах, проверка режима работы видеосистемы производилась динамически теми программами, которые с ней работали. Однако в данном случае это неприменимо, поскольку требует использования системных вызовов DOS. Вместо этого значение глобального указателя vid_mem устанавливается с помощью функции set_vid_mem, приводимой ниже.

```
set_vid_mem()
{
  int vmode;
  vmode = video_mode();
  if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
    printf("video must be in 80 column text mode");
    exit (1);

  }
  /* установить соответствующий адрес видеопамяти */
  if(vmode==7) vid_mem = (char far *) 0xB0000000;
```

```
  else vid_mem = (char far *) 0xB8000000;
}
```

Наконец, выход из функции main() осуществляется путем обращения к функции tsr(), приведенной ниже.

```
/* завершить выполнение, но оставить резидентной */
tsr(size)
unsigned size;
{
  union REGS r;
  r.h.ah = 49; /* завершить и оставить резидентной */
  r.h.al = 0;  /* код возврата */
  r.x.dx = size;
  int86(0x21, &r, &r);
}
```

Параметр size, определяемый в регистре DX, используется для того, чтобы сообщить DOS, сколько памяти требуется для размещения TSR-программы. Размер памяти определяется в 16-байтных параграфах. Иногда бывает трудно определить, сколько памяти необходимо для размещения программы. И если в этом случае вы разделите размер загрузочного модуля вашей программы (файла с расширением .EXE) на 16, а полученную величину умножите на 2, то будете застрахованы от ошибки. Точно определить размер необходимой памяти трудно, поскольку загрузочные модули частично накладываются друг на друга при загрузке и необязательно размещаются в непрерывной области. (Если вы намереватесь продавать свои программы, то наверняка хотели бы знать точно, сколько потребуется памяти, чтобы не оказаться слишком расточительным. Наиболее просто это можно определить экспериментальным путем). Код возврата, устанавливаемый в регистре AL, передается системе.

После завершения выполнения функции main() программа остается в памяти, и никакая другая программа не может быть загружена на ее место. Это значит, что прикладная часть программы в любой момент времени готова быть запущенной нажатием клавиши PT SCR.

Прикладная часть TSR-программы

Точкой входа в прикладную часть TSR-программы должна быть функция типа interrupt. В представленном ниже примере запуск прикладной части выполняется путем вызова функции window_main().

```
/* Точка входа в прикладную часть TSR-программы */
void interrupt tsr_ap()
{
  if(!busy) {
    busy = !busy;
    window_main();
    busy = !busy;
  }
}
```

Глобальная переменная busy первоначально устанавливается в 0. Прикладная часть TSR-программы не является повторно входимой, следовательно, она не должна запускаться дважды за время одного использования. Переменная busy используется как раз для того, чтобы предотвратить это. (Некоторые компиляторы Си могут создавать реентерабельные программы, но безопаснее для вас не обсуждать здесь этого вопроса).

В программы управления окнами необходимо внести некоторые изменения для того, чтобы их можно было использовать в TSR-программах. Во-первых, необходимо статически распределять память, необходимую для хранения текущего содержимого экрана, путем использования глобального массива. Вы могли привыкнуть к тому, что эта память распределялась динамически, но данный способ здесь непригоден, вследствие того, что функции динамического

распределения используют системный вызов, который недопустим в TSR-программах. По этой же причине функция go_to_xy() не может быть использована для позиционирования курсора. Наконец, стандартные Си-функции sscanf() и sprintf() также не могут быть использованы (по крайней мере, в Турбо Си), потому что также осуществляют обращения к DOS. Вместо них используются функции atoi() и itoa(). Полный текст программы резидентного калькулятора представлен ниже.

```
/* TSR-программа, использующая прерывание печати экрана */
#include "dos.h"
#include "stdlib.h"
#define BORDER 1
#define ESC 27
#define MAX_FRAME 1
#define REV_VID 0x70
#define NORM_VID 7
#define BKSP 8
void interrupt tsr_ap();
void save_video(), restore_video();
void write_string(), write_char();

void display_header(), draw_border();
void window_gets();
void window_cleol(), window();
void calc();
char far *vid_mem;
struct window_frame {
  int startx, endx, starty, endy;
  int curx, cury; /* текущее положение курсора в окне */
  unsigned char *p; /* указатель буфера */
  char *header;    /* сообщение в верхней части окна */
  int border; /* включение/отключение бордюра */
  int active; /* активация/деактивация окна */
} frame[MAX_FRAME];
char wp[4000]; /* буфер для хранения текущего содержимого экрана
/* busy установлена в 1, если программа активна, иначе - в 0 */
char busy = 0;
main()
{
  struct address {
    char far *p;
  } ;
  /* адрес прерывания печати экрана */
  struct address far *addr = (struct address far *) 20;
  addr->p = (char far *) tsr_ap;
  set_vid_mem();
  tsr(2000);
}
set_vid_mem()
{
  int vmode;
  vmode = video_mode();
  if((vmode!=2) && (vmode!=3) && (vmode!=7)) {
    printf("video must be in &0 column text mode");
    exit(1);
    }
    /* установить соответсвующий адрес видеопамяти */
    if(vmode==7) vid_mem = (char far *) 0xB0000000;
    else vid_mem = (char far *) 0xB8000000;
}
/* точка входа в прикладную часть TSR-программы */
void interrupt tsr_ap()
{
    if(!busy) {
```

```
      busy = !busy;
      window_main();
      busy = !busy;
    }
 }

 /* завершить, но оставить резидентной */
 tsr(size)
 unsigned size;
 {
   union REGS r;
   r.h.ah = 49;   /* завершить, но оставить резидентной */
   r.h.al = 0;    /* код возврата */
   r.x.ax = size;
   int86(0x21, &r, &r);
 }
   window_main()
 {
    /* первым делом, создать структуру окна */
    make_window(0, " Calculator ", 8, 20, 12, 60, BORDER);
    /* для активации описанного окна используйте window() */
    calc();
 }
/*************************************************************/
/* Функции управления окнами                                 */
/*************************************************************/
 /* Вывести на экран спускающееся меню */
 void window(num)
 int num; /* номер окна */
 {
   int vmode, choice;
   int x, y;
   /* сделать окно активным */
   if(!frame[num].active)  { /* используется не постоянно */
     save_video(num);        /* сохранить текущий экран */
     frame[num].active = 1;  /* установить флаг активности */
   }
   if(frame[num].border) draw_border(num);
   display_header(num); /* вывести окно */
 }
 /* Создать спускающееся окно
    если окно может быть создано, возвращается 1;
    иначе возвращается 0.
 */
 make_window(num, header, startx, starty, endx, endy, border)
 int num; /* номер окна */
 char *header;  /* текст заголовка */
  int startx, starty; /* координаты X,Y левого верхнего угла */
  int endx, endy; /* координаты X,Y правого нижнего угла */
  int border;  /* без бордюра если 0 */
  {

    register int i;
    int choice, vmode;
    unsigned char *p;
    if(num>MAX_FRAME)   {
      window_puts(0, "Too many windows\n");
      return 0;
 }
 if((startx>24) || (starty>78) || (starty<0))  {
   window_puts(0, "range error");
   return  0;
 }
 if((endx>24) || (endy>79))  {
```

```
  window_puts(0, "window won't fit");
  return 0;
}
/* создать структуру окна */
frame[num].startx = startx; frame[num].endx = endx;
frame[num].starty = starty; frame[num].endy = endy;
frame[num].p = wp;
frame[num].header = header;
frame[num].border = border;
frame[num].active = 0;
frame[num].curx = 0; frame[num].cury = 0;
return 1;
}
/* Деактивировать окно и удалить его с экрана */
deactivate(num)
int num;
{
  /* установить курсор в левый верхний угол */
  frame[num].curx = 0;
  frame[num].cury = 0;
  restore_video(num);
}
/* Вывести заголовок окна в соответсвующее поле */
void display_header(num)
int num;
{
  register int i, y, len;
  y = frame[num].starty;
  /* Вычислить точное значение центральной позиции заголовка
     если отрицательное - заголовок не может быть выведен
  */
   len = strlen(frame[num].header);
   len = (frame[num].endy - y - len) / 2;

   if(len<0) return; /* don't display it */
   y = y +len;
   write_string(frame[num].startx, y,
  frame[num].header, NORM_VID);
}
void draw_border(num)
int num;
{
  register int i;
  char far *v, far *t;
  v = vid_mem;
  t = v;
  for(i=frame[num].startx+1; i<frame[num].endx; i++) {
     v += (i*160) + frame[num].starty*2;
     *v++ = 179;
     *v = NORM_VID;
     v = t;
     v += (i*160) + frame[num].endy*2;
     *v++ = 179;
     *v = NORM_VID;
     v = t;
}
for(i=frame[num].starty+1; i<frame[num].endy; i++) {
   v += (frame[num].startx*160) + i*2;
   *v++ = 196;
   *v = NORM_VID;
   v = t;
   v += (frame[num].endx*160) + i*2;
   *v++ = 190;
   *v = NORM_VID;
```

```
   v = t;
 }
 write_char(frame[num].startx, frame[num].starty, 218, NORM_VID);
 write_char(frame[num].startx, frame[num].endy, 191, NORM_VID);
 write_char(frame[num].endx, frame[num].starty, 192, NORM_VID);
 write_char(frame[num].endx, frame[num].endy, 217, NORM_VID);
 }
 /*************************************************************/
 /*  Оконные функции ввода/вывода                             */
 /*************************************************************/
 /* Вывести строку начиная с текущей позиции курсора
    описанного окна.
    Возвратить 0 если окно не активное;
    и 1 в противном случае.
 */
 window_puts(num, str)
 int num;
 char *str;
 {

     /* убедиться, что окно активное */
    if(!frame[num].active) return 0;
for( ; *str; str++)
  window_putchar(num, *str);
return 1;
     }
     /* Вывести символ в текущую позицию курсора
        описанного окна.
        Возвратить 0 если окно не активное,
        и 1 в противном случае.
     */
     window_putchar(num, ch)
     int num;
     char ch;
     {
       register int x, y;
       char far *v;
       /* убедиться, что окно активное */
       if(!frame[num].active) return 0;
       x = frame[num].curx + frame[num].startx + 1;
       y = frame[num].cury + frame[num].starty + 1;
       v = vid_mem;
       v += (x*160) + y*2; /* вычислить адрес */
       if(y>=frame[num].endy) {
 return 1;
      }
      if(x>=frame[num].endx) {
 return 1;
     }
     if(ch=='\n') { /* символ перехода на новую строку */
     x++;
     y = frame[num].startx+1;
     v = vid_mem;
     v += (x+160) + y*2; /* вычислить адрес */
     frame[num].curx++;  /* инкрементировать X */
     frame[num].cury = 0; /* сбросить Y */
   }
   else {
     frame[num].cury++;
     *v++ = ch;  /* вывести символ */
     *v++ = NORM_VID; /* нормальные атрибуты символа */
   }
   window_xy(num, frame[num].curx, frame[num].cury);
   return 1;
```

```
  }
  /* Установка курсора в заданную позицию окна.
     Возвращает 0 при выходе за границу;
     не ноль в противном случае.

  */
  window_xy(num, x, y)
  int num, x, y;
  {
      if(x<0 || x+frame[num].startx>=frame[num].endx-1)
return 0;
      if(y<0 || y+frame[num].starty>=frame[num].endy-1)
return 0;
      frame[num].curx = x;
      frame[num].cury = y;
      return 1;
   }
   /* Считать строку из окна. */
   void window_gets(num, s)
   int num;
   char *s;
   {
     char ch, *temp;
     temp = s;
     for(;;) {
       ch = window_getche(num);
       switch(ch) {
 case '\r':  /* нажата клавиша ENTER */
   *s='\0';
   return;
 case BKSP: /* возврат */
   if(s>temp) {
     s--;
     frame[num].cury--;
     if(frame[num].cury<0) frame[num].cury = 0;
       window_xy(num, frame[num].curx, frame[num].cury);
 write_char(frame[num].startx+ frame[num].curx+1,
     frame[num].starty+frame[num].cury+1, ' ', NORM_VID);
   }
   break;
 default:  *s = ch;
  s++;
      }
    }
   }
   /* Ввод символа с клавиатуры в окно.
      Возвращает полный 16-разрядный скан-код.
   /*
   window_getche(num)
   int num;
   {
     union inkey {
       char ch[2];
       int i;
     } c;

     if(!frame[num].active) return 0; /* window not active */
     window_xy(num, frame[num].curx, frame[num].cury);
     c.i = bioskey(0);    /* обработать нажатие клавиши */
  if(c.ch[0]) {
   switch(c.ch[0]) {
      case '\r': /* нажата клавиша ENTER */
break;
      case BKSP: /* возврат */
```

```
        break;
      default:
if(frame[num].cury+frame[num].starty < frame[num].endy-1) {
write char(frame[num].startx+ frame[num].curx+1,
  frame[num].curx--;
window_xy(num, frame[num].curx, frame[num].cury);
      }
      return c.i;
   }
   /* Очистить до конца строки */
   void window_cleol(num)
   int num;
   {
     register int i, x, y;
     x = frame[num].curx;
     y = frame[num].cury;
     window_xy(num, frame[num].curx, frame[num].cury);
     for(i=frame[num].cury; i<frame[num].endy-1; i++)
       window_putchar(num,' ');
     window_xy(num, x, y);
  }
  /* Переместить курсор на одну строку вверх.
     При успешном завершении вернуть ненулевое значение;
     в противном случае - 0.
  */
  window_upline(num)
  int num;
  {
    if(frame[num].curx>0) {
      frame[num].curx--;
      window_xy(num, frame[num].curx, frame[num].cury);
      return 1;
   }
   return 0;
 }
 /* Переместить курсор на одну строку вниз.
    При успешном завершении вернуть ненулевое значение;
    в противном случае - 0.
 */
 window_downline(num)

int num;
{
  if(frame[num].curx<frame[num].endx-frame[num].startx-1} {
    frame[num].curx++;
    window_xy(num, frame[num].curx, frame[num].cury);
    return 1;
  }
  return 1;
}
/* вернуться на одну позицию назад */
window_bksp(num)
int num;
{
  if(frame[num].cury>0) {
    frame[num].cury--;
    window_xy(num, frame[num].curx, frame[num].cury);
    window_putchar(num, ' ');
    frame[num].cury--;
    window_xy(num, frame{num}.curx, frame[num].cury);
  }
}
/*******************************************************/
/* Дополнительные функции                              */
```

```
/*******************************************************/
/* Вывести строку с установленными атрибутами */
void write_string(x, y, p, attrib)
int x, y;
char *p;
int attrib;
{
  register int i;
  char far *v;
  v = vid_mem;
  v += (x*160) + y*2; /* вычислить адрес */
  for(i+y; *p; i++)  {
    *v++ = *p++;  /* вывести символ */
    *v++ = attrib;    /* вывести атрибуты */
  }
}
/* Вывести символ с утановленными атрибутами */
void write_char(x, y, ch, attrib)
int x, y;
char ch;
int attrib;
{
  register int i;
  char far *v;
  v = vid_mem;

  v += (x*160) +y*2;
  *v++ = ch;  /* вывести символ */
  *v = attrib;    /* вывести атрибуты */
}
/* Сохранить содержимое области экрана */
void save_video(num)
int num;
{
  register int i,j;
  char far *v, far *t;
  char *but_ptr;
  but_ptr = frame[num].p;
  v = vid_mem;
  t=v;
  for(i=frame[num].starty; i<frame[num].endy+1; i++)
    for(j=frame[num].startx; j<frame[num].endx+1; j++) {
      t = (v + (j*160) + i*2);
      *buf_ptr++ = *t++;
      *buf_ptr++ = *t;
      *(t-1) = ' ';  /* очистить окно */
    }
}
/* Восстановить содержимое области экрана */
void save_video(num)
int num;
{
  register int i,j;
  char far *v, far *t;
  char *but_ptr;
  but_ptr = frame[num].p;
  v = vid_mem;
  t=v;
  for(i=frame[num].starty; i<frame[num].endy+1; i++)
    for(j=frame[num].startx; j<frame[num].endx+1; j++) {
      v = t;
      v += (j*160) + i*2;
      *v++ = *but_ptr++;   /* вывести символ */
      *v = *but_ptr++;   /* вывести атрибуты */
```

```
}
frame[num].active = 0; /* восстановить изображение */
     }
     /* Возвращает код текущего видеорежима */
     video_mode()
     {
union REGS r;
r.h.ah =15;  /* получить код видеорежима */
return int86(0x10, &r, &r) & 255;
}

/*********************************************************
         калькулятор
*********************************************************/
      #define MAX 100
      int *p;   /* указатель стека */
      int *tos; /* указатель вершины стека  */
      int *bos; /* указатель дна стека */
    char in[80], out[80];
    int stack[MAX];
    /* Стековый, с постфиксной записью калькулятор
    на четыре функции */
    void calc()
    {
      int answer;
      int a, b;
      p = stack;
      tos = p;
      bos = p + MAX - 1;
      window(0);
      do {
        window_xy(0, 0, 0);
        window_cleol(0);
        window_puts(0, " : "); /* промптер калькулятора */
        window_gets(0, in);
        window_puts(0, " \n ");
        window_cleol(0);
        switch(*in) {
          case '+ ':
            a = pop();
            b = pop();
            answer = a + b;
            push(a+b);
            break;
  case '-':
            a = pop();
            b = pop();
            answer = b-a;
            push(b-a);
            break;
          case '- ':
          a = pop();
          b = pop();
          answer = b*a;
          push(b*a);
          break;
        case '/ ':
          a = pop();
          b = pop();

          if(a==0) {
              window_puts(0, "divide by 0\n");
              break;
          }
```

```
            answer = b/a;
            push(b/a);
            break;
            default:
              push(atoi(in));
              continue;
        }
        itoa(answer, out, 10);
        window_puts(0, out);
    }   while(*in);
    deactivate(0);
}
/* Поместить число в стек.
   Возвратить 1 при успешном завершении;
   и 0, если стек переполнен
*/
push(i)
int i;
{
  if(p>bos) return 0;
  *p = i;
  p++;
  return 1;
}
/* Извлечь верхний элемент стека
   Возвратить 0, если стек пуст.
*/
pop()
{
  p--;
  if(p<tos) {
    p++;
    return 0;
  }
  return *p;
}
```

Вы можете сразу вводить эту программу в ЭВМ. Для того, чтобы установить прикладную часть, запустите ее на выполнение. Для вызова калькулятора нажмите клавишу PT SCR.

Использование прерывания по нажатию клавиши.

Прерывание печати экрана очень просто использовать, но у него есть три крупных недостатка. Во-первых, оно позволяет быть резидентным в системе только прикладной части TSR-программы. Во-вторых, вы не можете при этом пользоваться печатью экрана. В-третьих, это решение проблемы "в лоб", и потому оно не очень хорошее. Лучшим способом запуска TSR-программы является использование прерывания 9 по нажатию клавиши. Прерывание 9 выполняется при каждом нажатии клавиши на клавиатуре.

При использовании прерывания 9 для запуска TSR-программ должны соблюдаться следующие основные положения. Во-первых, Вы должны переписать адрес из таблицы векторов, соответствующий прерыванию 9, в такое место таблицы, которое соответствует неиспользуемому DOS прерыванию. Мы будем использовать прерывание 60. Затем, занесите адрес точки входа в вашу TSR-программу по адресу прерывания 9 в таблице векторов. После запуска ваша TSR-программа первым делом вызовет через прерывание драйвер ввода с клавиатуры. Затем проверяется, не соответствует ли введенный символ "горячей клавише", которая используется для запуска прикладной части TSR-программы. Если соответствует, то прикладная часть начинает выполняться, в противном случае никакого действия не производится и TSR-программа деактивируется. Таким образом, при каждом нажатии происходит обращение к функции, реагирующей на

нажатие клавиш, но прикладная часть TSR-программы запускается только при нажатии определенной клавиши.

Использование прерывания по нажатию клавиши имеет два преимущества. Во-первых, при этом нет никакой потери функциональных возможностей. Во-вторых, появляется возможность использовать одновременно несколько различных прикладных частей TSR-программы, вызов которых осуществляется нажатием соответствующих им различных "горячих клавиш". Представленная в данном разделе TSR-программа использует эту возможность и включает в свой состав и "калькулятор", и "записную книжку" (из раздела 2), которые вызываются отдельно друг от друга.

Прежде, чем использовать эту возможность, вы должны узнать кое-что об обработке BIOS нажатий клавиш.

Буфер символов, введенных с клавиатуры.

Как вы знаете, стандартные версии DOS буферизуют до 15 символов, введенных с клавиатуры, что позволяет выполнить ввод с опережением. При каждом нажатии клавиши наступает прерывание 9. Программа ISR реакции на нажатие клавиши принимает код символа из порта и помещает его в буфер. Когда вы обращаетесь к функциям DOS или BIOS ввода с клавиатуры, обрабатывается только содержимое буфера, а не текущее содержимое порта. Это позволяет вашим программам непосредственно обрабатывать содержимое буфера символов, так же, как это делают программы BIOS и DOS. Таким образом, это позволяет функции реагирования на нажатие клавиш вашей TSR-программы определять, была ли нажата "горячая клавиша", не уничтожая при этом содержимого буфера символов.

Буфер ввода с клавиатуры расположен по адресу 0000:041 (1054 в десятичной системе счисления). Поскольку при каждом нажатии клавиши формируется 16-битный скан-код, то для ввода 15 символов требуется 30 байт. Однако обычно используются 32 байта, т.к. скан-код клавиши **RETURN** автоматически добавляется к концу буфера. Буфер организован в виде циклической очереди, доступ к которой осуществляется через указатели начала и конца очереди. Указатель начала указывает на символ, который был введен последним. Указатель конца указывает на следующий символ, который будет передан по запросу на ввод символа от DOS или BIOS. Указатель начала хранится по адресу 0000:041C (1052 в десятичной с.с.). Значения указателей начала и конца фактически используются для индексной адресации очереди, и соответствует индексу текущей позиции +30. (Это связано с особенностями выполнения косвенной адресации процессором 8086). Значения указателей начала и конца очереди совпадают в том случае, если очередь пуста.

Функция инициализации.

Для прикладной TSR-программы, представленной в данном разделе, требуется небольшая по объему программа инициализации. Она оформлена в виде функции main(), которая приводится ниже.

```
main()
{
  struct address {
    char far *p;
  } temp;
  /* указатель вектора прерывания 9 */
  struct address far *addr = (struct address far *) 36;
  /* указатель вектора прерывания 60 */
  struct address far *int9 = (struct address far *) 240;
  /* Поместить адрес обработки прерывания от клавиатуры
     в вектор прерывания 60. Если вектора прерываний 60 и
     61 содержат одинаковые адреса, то TSR-программа не
     была запущена.
  */
```

```
  if(int9->p == (int9+1)->p) {
    int9->p = addr->p;
    addr->p = (char far *) tsr_ap;
  printf("tsr installed - F2 for note pad, F3 for calculator ");
  } else {
    printf ("tsr application already initialized\n ");
    exit(1);
}  }
  set_vid_mem();
  tsr(2000);
}
```

Следует отметить, что данная версия программы не допускает, чтобы ее запускали более одного раза в течение одного сеанса работы. Это связано с тем, что повторный запуск программы приведет к записи адреса точки входа в TSR-программу в таблицу векторов по адресу 60-го прерывания, а содержавшийся там адрес программы реакции на нажатие клавиши будет запорчен. Во время работы функции проверяется, совпадает ли содержимое таблицы векторов, соответствующее прерываниям 60 и 61. (Прерывание 61 также не используется DOS). DOS обрабатывает все неиспользуемые ею прерывания одной и той же программой обработки недопустимого прерывания. Следовательно, перед запуском TSR-программы эти адреса будут совпадать, а после запуска они будут различны.

Прикладная часть TSR-программы.

Применяемая здесь функция входа в TSR-программу является более сложной, чем при использовании прерывания по печати экрана. Первым делом она должна сформировать прерывание 60, для того чтобы ввод с клавиатуры осуществлялся стандартной программой ввода. Большинство компиляторов Си имеют функцию генерации прерывания. В Турбо Си это функция gen**int**errupt(), параметром которой является номер того прерывания, которое вы хотите вызвать. После возврата из прерывания 60 ваша функция должна проверить содержимое очереди, адресуемое с помощью указателя начала, на предмет того, не была ли нажата "горячая клавиша". Для представленной здесь программы "горячими" являются клавиши F2 и F3 с позиционными кодами 60 и 61 соответственно. Нажатие "горячей клавиши" должно быть обнаружено прежде, чем управление будет передано прикладной части программы. Глобальная переменная busy используется для того, чтобы предотвратить одновременное использование обеих прикладных частей программы, поскольку большинство компиляторов Си не позволяют создавать повторно-входимые программы. Если одна из прикладных частей активна, то другой части активация в этот момент запрещена. Функция tsr_ap() приводится ниже.

```
/* Точка входа в прикладную часть TSR-программы */
void interrupt tsr_ap()
{
  char far *t = (char far *) 1050; /* адрес указателя заголовка *
  geninterrupt(60);
  if(*t != *(t+2)) { /* если не пустой */
    t += *t - 30 + 5; /* перейти к позиции символа */
    if(*t == 60 || *t == 61) {
      bioskey(0); /* сбросить клавиши F2/F5 */
      if(!busy) {
        busy = !busy;
      }
    }
  }
}
```

Следует отметить, что параметром функции window_main() является позиционный код "горячей клавиши", для того, чтобы она могла осуществить выбор соответствующей прикладной части.

```
/* создать окно */
window_main(which)
int which;
{
  union inkey {
    char ch[2];
    int i;
  } c;
  int i;
  char ch;

  /* во-первых, создать рамку окна */
  make_window(0," Notepad [F1 to exit] ", 5, 20, 17, 60, BORDER);
  make_window(1, " Colculator ", 8, 20, 12, 60, BORDER);
  /* использовать window() для активации созданного окна */
  switch(which) {
    case 60:
      notepad();
      break;
    case 61:
      calc();
      break;
  }
}
```

Вы можете сразу вводить в ЭВМ представленную здесь программу. После того, как вы ее запустите, клавишей F2 будет выбираться программа "записная книжка", а клавишей F3 - "калькулятор".

```
/* Программа, остающаяся резидентной после завершения и
   использующая прерывание 9 от клавиатуры.
*/
#include "dos.h "
#include "stdlib.h "
#include "ctype.h "
#define BORDER 1
#define ESC 27
#define MAX_FRAME 2
#define REV_VID 0x70
#define NORM_VID 7
#define BKSP 8
void interrupt tsr_ap();
void save_video(), restore_video();
void write_string(), write_char();
void display_header(), draw_border();
void window_gets();
void window_cleol(), window();
void notepad(), calc();
char far *vid_mem;
char wp[4000]; /* буфер для хранения текущего
                  содержимого экрана */
struct window_trame {
  int startx, endx, starty, endy;
  int curx, cury; /* текущее положение курсора в окне */
  unsigned char *p; /* указатель в буфере */
  char *header; /* сообщение заголовка */
  int border; /* включение/отключение бордюра */
  int active; /* выводить/не выводить на экран */
} frame [MAX_FRAME];

char in[80], out[80];
/* busy установлена в 1,когда программа активна, иначе - в 0 */
char busy = 0;
main()
{
```

```
struct adaress {
  char far *p;
  }temp;
/* указатель на вектор прерывания 9 */
struct address far *addr = (struct address far *) 36;
/* указатель на вектор прерывания 60 */
struct address far *int9 = (struct address far *) 240;
/* Поместить адрес программы обработки прерывания от клавиатуры
   по адресу прерывания 60. Если вектора прерываний 60 и 61
   содержат одинаковые адреса, то TSR-программа не была запущена.
*/
if(int9->p == (int9+1)->p) {
int9->p = addr->p;
addr->p = (char far *) tsr_ap;
printf ("tsr installed - F2 for note pad, F3 for calculator");
} else {
printf ("tsr application already initialized\n");
exit (1);
}
set_vid_mem();
tsr (800);
}
set_vid_mem()
{
  int vmode;
  vmode = video_mode();
  if(( vmode != 2) && ( vmode != 3) && ( vmode != 7))  {
    printf("video must be in 80 column text mode");
    exit(1);
  }
  /* установить соответствующий адрес видеопамяти */
  if(vmode==7) vid_mem = (char far *) 0xB0000000;
  else vid_mem = (char far *) 0xB8000000;
}

/* Точка входа в прикладную часть TSR-программы */
void interrupt tsr_ap()
{
  char far *t = (char far *) 1050; /* адрес указателя заголовка *
  genterrupt(60);/* читать символ */
  if(*t != *(t+2)) { /* если не пусто */
  t += *t-30+5; /* перейти к позиции символа */
  if(*t == 60 || *t == 61)  {
  bioskey(0); /* сбросить клавиши F2/F3 */
  if(!busy) {
    busy = !busy;

    window_main(*t);
    busy = !busy;
   }
  }
 }
}
/* завершить но оставить резидентной */
tsr(size)
unsigned size;
{
  union REGS r;
  r.h.ah = 49;  /* завершить и оставить резидентной */
  r.h.al = 0;  /* код возврата */
  r.x.dx = size;  /* размер программы/16 */
  int86(0x21, &r, &r);
}
/* создать окно */
```

```
window_main(which)
int which;
{
  union inkey  {
    char ch[2];
    int i;
  } c;
  int i;
  char ch;
  /* во-первых, создать рамку окна */
 make_window(0, " Notepad [F1 to exit] ", 5, 20, 17, 60, BORDER);
 make_window(1, " Calculator ", 8, 20, 12, 60, BORDER);
  /* использовать window() для активации созданного окна */
  switch(which)  {
    case 60:
      notepad();
      break;
      case 61:
        calc();
        break;
   }
}
/*************************************************************/
/* Функции управления окнами                                 */
/*************************************************************/
/* Вывести спускающееся окно */
void window(num)
int num; /* номер окна */
{
  /* сделать окно активным */

  if(!frame[num].active)  {  /* используется не постоянно */
    save_video(num);         /* сохранить текущий экран */
    frame[num].active = 1;  /* установить флаг активности */
    }
    if(frame[num].border) draw_border(num);
    display_header(num); /* вывести окно */
}
/* Создать рамку спускающегося окна.
   Если рамка может быть создана, возвращается 1,
   в противном случае возвращается 0.
*/
make_window(num, header, startx, starty, endx, endy, border)
int num;     /* номер окна */
char *header;   /* текст заголовка */
int startx, starty;  /* координаты X,Y верхнего левого угла */
int endx, endy;  /* координаты X,Y нижнего правого угла */
int border;     /* без бордюра если 0 */
{
  register int i;
  int choice, vmode;
  unsigned char *p;
  if(num>MAX_FRAME)  {
    window_puts(0, "Too many windows\n");
    return 0;
  }
  if((startx>24) || (startx<0) || (starty>78) || (starty<0)) {
    window_puts(0, "range error");
    return 0;
  }
  if((endx>24) || (endy>79)) {
    window_puts(0, "window won't fit");
    return 0;
  }
```

```
/* Отвести достаточное количество памяти */
p= (unsigned char *) malloc(2*(endx-startx+1)*(endy-starty+1));
  if(!p) exit(1); /* поместите здесь ваш собственный
                     обработчик ошибок */
  /* создать рамку */
  frame[num].startx = startx; frame[num].endx = endx;
  frame[num].starty = starty; frame[num].endy = endy;
  frame[num].p = wp;
  frame[num].header = header;
  frame[num].border = border;
  frame[num].active = 0;
  frame[num].curx = 0;  frame[num].cury = 0;
  return 1;
}

/* Деактивировать окно и удалить его с экрана */
deactivate(num)
int num;
{
   /* установить курсор в левый верхний угол */
   frame[num].curx = 0;
   frame[num].cury = 0;
   restore_video(num);
}
/* Вывести текст заголовка в соответствующее поле */
void display_header(num)
int num;
{
  register int i, y, len;
  y = frame[num].starty;
/* Вычислить начальную позицию относительно
   центра заголовка, если отрицательная, то
   сообщение не подходит.
*/
len = strlen(frame[num].header);
len = (frame[num].endy - y - len) / 2;
if(len<0) return; /* не выводить его */
y = y +len;
write_string(frame[num].startx, y,
             frame[num].header, NORM_VID);
{
void draw_border(num)
int num;
{
  register int i;
  char far *v, far *t;
  v = vid_mem;
  t = v;
  for(i=frame[num].startx+1; i<frame[num].endx; i++) {
     v += (i*160) + frame[num].starty*2;
     *v++ = 179;
   *v = NORM_VID;
    v = t;
    v += (i*160) + frame[num].endy*2;
    *v++ = 179;
    *v = NORM_VID;
    v = t;
}
for(i=frame[num].starty+1; i<frame[num].endy; i++) {
   v += (frame[num].startx*160) + i*2;
   *v++ = 196;
   *v = NORM_VID;
   v = t;
   v += (frame[num].endx*160) + i*2;
```

```
   *v++ = 196;
   *v = NORM_VID;

   v = t;
}
write_char(frame[num].startx, frame[num].starty, 218, NORM_VID);
write_char(frame[num].startx, frame[num].endy, 191, NORM_VID);
write_char(frame[num].endx, frame[num].starty, 192, NORM_VID);
write_char(frame[num].endx, frame[num].endy, 217, NORM_VID);
}
/*************************************************************/
/* Оконные функции ввода/вывода                              */
/*************************************************************/
/* Вывести строку начиная с текущей позиции
   в созданном окне.
   Возвратить 0, если окно не активное,
   и 1 - в противном случае.
*/
window_puts(num, str)
int num;
char *str;
{
   /* убедитесь, что окно активное */
  if(!frame[num].active) return 0;
  for( ; *str; str++)
    window_putchar(num, *str);
  return 1;
}
/* Вывести символ в текущую позицию курсора
   в созданном окне
   Возвратить 0, если окно не активное,
   и 1 - в противном случае.
*/
window_putchar(num, ch)
int num;
char ch;
{
  register int x, y;
  char far *v;
   /* убедитесь, что окно активное */
  if(!frame[num].active) return 0;
  x = frame[num].curx + frame[num].startx + 1;
  y = frame[num].cury + frame[num].starty + 1;
  v = vid_mem;
  v += (x*160) + y*2; /* вычислить адрес */
  if(y>=frame[num].endy) {
    return 1;
  }
  if(x>=frame[num].endx) {
    return 1;
  }

  if(ch=='\n') { /* символ перехода к новой строке */
    x++;
    y = frame[num].startx+1;
    v = vid_mem;
    v += (x*160) + y*2; /* вычислить адрес */
    frame[num].curx++; /* нарастить X */
    frame[num].cury = 0; /* сбросить Y */
  }
  else {
    frame[num].cury++;
    *v++ = ch; /* вывести символ */
    *v++ = NORM_VID; /* нормальные атрибуты символа */
```

```
  }
  window_xy(num, frame[num].curx, frame[num].cury);
  return 1;
}
/* Установить курсор в определенной позиции окна.
   Возвратить 0 при выходе за границу;
   и не ноль в противном случае.
*/
window_xy(num, x, y)
int num, x, y;
{
  if(x<0 || x+frame[num].startx>=frame[num].endx-1)
    return 0;
  if(y<0 || y+frame[num].starty>=frame[num].endy-1)
    return 0;
  frame[num].curx = x;
  frame[num].cury = y;
  return 1;
}
/* Считать строку из окна */
void window_gets(num, s)
int num;
char *s;
{
  char ch, *temp;
  char out[10];
     temp = s;
     for(;;) {
       ch = window_getche(num);
       switch(ch) {
         case '\r': /* нажата клавиша ENTER */
           *s='\0';
           return;
         case BKSP: /* возврат на шаг */
           if(s>temp) {
           s--;
           frame[num].cury--;
           if(frame[num].cury<0) frame[num].cury = 0;

             window_xy(num, frame[num].curx, frame[num].cury);
               write_char(frame[num].startx+ frame[num].curx+1,
             frame[num].starty+frame[num].cury+1,' ',NORM_VID);
           }
           break;
         default: *s = ch;
           s++;
     }
   }
}
/* Ввести символ в окно.
   Возвратить полный 16-разрядный скан-код
*/
window_getche(num)
int num;
{
  union inkey {
    char ch[2];
    int i;
  } c;
  if(!frame[num].active) return 0; /* окно не активное */
  window_xy(num, frame[num].curx, frame[num].cury);
  c.i = bioskey(0);  /* обработать нажатие клавиши */
  if(c.ch[0]) {
    switch(c.ch[0]) {
```

```
      case'\r': /* нажата клавиша ENTER */
        break;
      case BKSP: /* возврат на шаг */
        break;
      default:
     if(frame[num].cury+frame[num].starty < frame[num].endy-1) {
     write_char(frame[num].startx+ frame[num].curx+1,
       frame[num].starty+frame[num].cury+1, c.ch[0], NORM_VID);
       frame[num].cury++;
        }
   }
   if(frame[num].curx < 0) frame[num].curx = 0;
   if(frame[num].curx+frame[num].startx > frame[num].endx-2)
   frame[num].curx--;
      window_xy(num, frame[num].curx, frame[num].cury);
    }
    return c.i;
}
/* Очистить до конца строки */
void window_cleol(num)
int num;
{
  register int i, x, y;
  x = frame[num].curx;
  y = frame[num].cury;
  window_xy(num, frame[num].curx, frame[num].cury);

  for(i=frame[num].cury; i<frame[num].endy-1; i++)
    window_putchar(num,' ');
  window_xy(num, x, y);
}
/* Переместить курсор вверх на одну строку.
   Возвратить не ноль в случае успеха
   и 0 в противном случае.
*/
window_upline(num)
int num;
{
  if(frame[num].curx>0) {
    frame[num].curx--;
    window_xy(num, frame[num].curx, frame[num].cury);
    return 1;
  }
  return 0;
}
/* Переместить курсор вниз на одну строку.
   Возвратить не ноль в случае успеха
   и 0 в противном случае.
*/
window_downline(num)
int num;
{
if(frame[num].curx<frame[num].endx-frame[num].startx-1) }
  frame[num].curx++;
  window_xy(num, frame[num].curx, frame[num].cury);
  return 1;
 }
 return 1;
}
/* назад на один символ */
window_bksp(num)
int num;
{
if(frame[num].cury>0) {
```

```
  frame[num].cury--;
  window_xy(num, frame[num].curx, frame[num].cury);
  window_putchar(num, ' ');
     frame[num].cury--;
     window_xy(num, frame[num].curx, frame[num].cury);
   }
}
/*************************************************************/
/* Дополнительные функции                                    */
/*************************************************************/
/* Вывести строку с указанными атрибутами */

void write_string(x, y, p, attrib)
int x, y;
char *p;
int attrib;
{
  register int i;
  char far *v;
  v=vid_mem;
  v += (x*160) + y*2; /* вычислить адрес */
  for(i=y; *p; i++) {
    *v++ = *p++;  /* записать символ */
    *v++ = attrib;  /* записать атрибуты */
  }
}
/* Вывести символ с указанными атрибутами */
void write_char(x, y, ch, attrib)
int x, y;
char ch;
int attrib;
{
  register int i;
  char far *v;
  v = vid_mem;
  v += (x*160) + y*2;
  *v++ = ch;  /* записать символ */
  *v = attrib;  /* записать атрибуты */
}
/* Сохранить область экрана */
void save_video(num)
int num;
{
  register int i, j;
  char *buf_ptr;
  char far *v, far *t;
  buf_ptr = frame[num].p;
  v = vid_mem;
  for(i=frame[num].starty; i<frame[num].endy+1; i++)
    for(j=frame[num].startx; j<frame[num].endx+1; j++) {
      t = (v + (j*160) + i*2);
      *buf_ptr++ = *t++;
      *buf_ptr++ = *t;
      *(t-1) = ' '; /* очистить окно */
    }
}
/* Восстановить область экрана */
void restore_video(num)
int num;
{
  register int i,j;

  char far *v, far *t;
  char *buf_ptr;
```

```
  buf_ptr = frame[num].p;
  v = vid_mem;
  t = v;
  for(i=frame[num].starty; i<frame[num].endy+1; i++)
    for(j=frame[num].startx; j<frame[num].endx+1; j++) {
      v = t;
      v += (j*160) + i*2;
      *v++ = *buf_ptr++; /* записать симпол */
      *v = *buf_ptr++;  /* записать атрибуты */
  }
  frame[num].active = 0;/* восстановить изображение */
}
/* Возвратить код текущего видеорежима */
video_mode()
{
  union REGS r;
  r.h.ah = 15;  /* получить код видеорежима */
  return int86(0x10, &r, &r) & 255;
}
/*************************************************/
/* Функции всплывающих окон                      */
/*************************************************/
#define MAX 100
int *p; /* указатель стека */
int *tos; /* указатель вершины стека */
int *bos; /* указатель дна стека */
int stack[MAX];
/* Стековый, с постфиксной записью калькулятор
с четырьмя функциями
*/
void calc()
{
  int answer;
  int a,b;
  p = stack;
  tos = p;
  bos = p+MAX-1;
       window(1);
       do {
         window_xy(1, 0, 0);
         window_cleol(1);
         window_puts(1, ": "); /* промптер калькулятора */
         window_gets(1, in);
         window_puts(1, "\n ");
         window_cleol(1);
         switch(*in) {


           case '+':
             a = pop();
             b = pop();
             answer = a+b;
             push(a+b);
             break;
             case '-':
               a = pop();
               b = pop();
               answer = b-a;
               push(b-a);
               break;
             case '* ':
               a = pop();
               b = pop();
               answer = b*a;
```

```
                push(b*a);
                break;
              case '/ ':
                a = pop();
                b = pop();
                if(a==0) {
                    window_puts(0, "divide by 0\n");
                    break;
                }
                answer = b/a;
                push(b/a);
                break;
              default:
                push(atoi(in));
                continue;
            }
            itoa(answer, out, 10);
            window_puts(1, out);
        }   while(*in);
        deactivate(1);
}
/* Поместить число в стек.
   Возвратить 1 в случае успеха
   и 0 если стек переполнен
*/
push(i)
int i;
{
  if(p>bos) return 0;
  *p = i;
  p++;
   return 1;
}
/* Извлечь верхний элемент из стека.
   Возвратить 0 если стек пуст.
*/
pop()


{
  p--;
  if(p<tos) {
    p++;
    return 0;
  }
  return *p;
}
/*********************************************************/
/*                Всплывающая записная книжка            */
#define MAX_NOTE 10
#define BKSP 8
char notes[MAX_NOTE] [80];
void notepad()
{
  static first = 1;
  register int i, j;
  union inkey {
    char ch[2];
    int i;
  } c;
  char ch;
  /* инициализировать массив записей если это необходимо */
  if(first) {
    for(i=0; i<MAX_NOTE; i++)
```

```
      *notes[i] = '\0 ';
    first = !first;
    }
    window(0);
  /* вывести существующие записи */
    for(i=0; i<MAX_NOTE; i++) {
      if(*notes[i]) window_puts(0, notes[i]);
      window_putchar(0, '\n ');
      }
      i = 0;
      window_xy(0, 0, 0);
      for(;;) {
        c.i = bioskey(0);  /* обработать нажатие клавиши */
        if(tolower(c.ch[1])==59) { /* F1 для выхода */
          deactivate(0);
          break;
        }
        /* если обычная клавиша */
        if(isprint(c.ch[0]) || c.ch[0]==BKSP) {
          window_cleol(0);
          notes[i][0] = c.ch[0];
          j = 1;
          window_putchar(0, notes[i][0]);
          do {


            ch = window_getche(0);
            if(ch == BKSP) {
              if( j>0 ) {
              j--;
              window_bksp(0);
            }
          }
          else {
            notes[i][j] = ch;
            j++;
          }
        } while(notes[i][j-1]! = '\r ');
        notes[i][j-1] = '\0 ';
        i++;
        window_putchar(0, '\n ');
      }
      else { /* если специальная клавиша */
        switch(c.ch[1]) {
          case 72: /* стрелка вверх */
            if(i>0) {
              i--;
              window_upline(0);
            }
           break;
         case 80: /* стрелка вниз */
           if(i<MAX_NOTE-1) {
             i++;
             window_downline(0);
           }
           break;
        }
     }
  }
}
```

Есть одно средство DOS, которое не описывается в документации, и которое может сделать TSR-программы более надежными в тот период времени, когда они используют много системных ресурсов. Вообще говоря, если прикладная часть вашей TSR-программы занимается обменом только с консолью, то вы застрахованы от неприятностей. Неприятности могут возникнуть при использовании таких объектов, как дисковые файлы или порты ввода-вывода. Хотя это и не описано в технических руководствах по операционной системе, но DOS вызывает прерывание 28H, когда она находится в "безопасном", т.е. холостом состоянии. Как вам наверное известно, при выполнении определенных функций DOS, которые относятся к критическим участкам, после начала их выполнения прерывания должны быть запрещены. Прерывание 28H никогда не вызывается DOS во время выполнения критического участка. Вы можете использовать это средство для защиты от сбоев вашей TSR-программы. Хотя здесь не представлено никаких примеров программ, но предлагаются следующие общие соображения по этому вопросу.

Главное отличие, которое вы должны иметь ввиду при использовании прерывания 28H, заключается в способе активации прикладной части вашей TSR-программы. Когда вызывается прерывание 28H, прикладная часть TSR-программы не может больше активироваться через программу обработки нажатий клавиш. Вместо этого программа обработки нажатий клавиш при нажатии "горячей клавиши" просто устанавливает флаг (в дальнейшем именуемый is-hotkey). Перед тем, как прикладная часть вашей TSR-программы может быть вызвана, вы должны создать новый обработчик прерывания 28H, который будет проверять, установлен флаг is-hotkey или нет. Если установлен, то прикладная часть активируется, сбрасывая при этом флаг is-hotkey. При этом вы обязательно должны не просто изменить первоначальное содержимое вектора прерывания 28H, а напротив, сохранить его и вызывать исходное прерывание 28H из вашего обработчика 28-го прерывания.

Если вы собираетесь продавать ваши TSR-программы, то должны обязательно использовать прерывание 28H (хотя по нему и нет документации), поскольку оно позволяет избежать случайных прерываний DOS во время выполнения критических участков.

Проблемы при создании TSR-программ

TSR-программы по своей природе очень склонны к сбоям. Например, использование TSR-программы, разработанной одним программистом, часто делает невозможным одновременное использование другой TSR-программы, разработанной другим программистом, поскольку обе программы будут пытаться переназначить адрес программы обработки прерывания 9 в таблице векторов на себя. Несомненно, при использовании своих TSR-программ вы можете избежать такого рода проблем, но будьте внимательны, если в вашей системе присутствует чужая TSR-программа.

ГЛАВА 4

ГРАФИКА

В этой главе приводится базовый набор функций графики, которые позволяют рисовать (отображать на экране) точки, линии, прямоугольники, окружности, используя возможности графических адаптеров CGA или EGA. Эти программы используются как основа, на которой строятся функции графики более высокого уровня. Для более детального изучения этих базовых графических функций предлагается книга "C: The Complete Reference" Herb Schild (Osborn / McGrow-Hall, 1987).

Помимо краткого представления базовых функций графики, в

этой главе приводятся следующие программы:

- сохранение графических изображений в файле на диске;
- загрузка графических изображений из файла;
- вращение объектов в двумерном пространстве;
- копирование или пересылка графических изображений.

В конце главы привевен текст "программы-художника", позволяющей рисовать на экране терминала, с использованием клавиш перемещения курсора.

По мере возрастания вашего опыта по использованию функций графики, вы сможете самостоятельно разрабатывать хорошие программы. Например, используя функции сохранения и загрузки графических изображений, вы сможете создавать графики или диаграммы и успешно использовать их в случае необходимости. Используя функции вращения изображений объектов, вы сможете разрабатывать программы "звездных войн" - образец "живой" графики, которые будут представлять большой интерес для вас. Вы сможете также, использовать эти функции как основу для использования систем автоматизированного проектирования (CAD/CAM system).

Для корректного использования программ, описаных в этой главе, вам необходимы компьютеры типа IBM PC XT, IBM PC AT или другие совместимые с ними модели, снабженные графическими адаптерами CGA или EGA. Все программы, приведенные в данной главе, кроме программ изображения точки, аппаратно-независимы, и вы можете с минимальными усилиями сделать их рабочими на других типах графических машин.

ВИДЕОРЕЖИМЫ И ЦВЕТОВАЯ ПАЛИТРА

Перед использованием каких-либо функций графики компьютер должен быть переведен в соответствующий видеорежим. Для компьютеров типа IBM PC это означает, что должны быть выбраны подходящие режим и палитра.

В таблице 4-1 приведены различные видеорежимы, в которых могут работать компьютеры IBM PC. Для функций, приведенных в этой главе, требуется 4 видеорежим, предполагающий использование цветного графического дисплея с размерностью экрана 320 на 200. Хотя адаптер EGA поддерживает и режимы с расширенной разрешающей способностью дисплея, 4 видеорежим выбран в качестве базового для разработки и использования функций графики в связи с тем, что он поддерживается как адаптером EGA, так и CGA. Использование особых режимов EGA требует только изменения функций записи точки (смотри книгу по использованию графики в EGA "Advance Grafics in C" Nelson Jobson Osborn/McGrow-Hall, 1987). Вам необходимо запомнить, что во всех кодах верхний левый угол имеет координаты 0,0.

BIOS-прерывание 16, функция 0, устанавливает видеорежим и используется в функции mode(), текст которой приведен ниже. Но прежде чем вы ознакомитесь с ней, предлагаем вам внимательно изучить все видеорежимы, поддерживаемые соответствующими адаптерами. Для этого обратитесь к таблице 4-1.

Таблица 4-1 Режимы терминала для машин IBM PC

Режим	Тип	Размерность экрана	Адаптер
0	алфавитно-цифровой, ч/б	40x25	CGA, EGA
1	алфавитно-цифровой, 16 цв.	40x25	CGA, EGA
2	алфавитно-цифровой, ч/б	80x25	CGA, EGA
3	алфавитно-цифровой, 16 цв.	80x25	CGA, EGA
4	графический, 4 цвета	320x200	CGA, EGA
5	графический, 4 серых тона	320x200	CGA, EGA
6	графический, ч/б	640x200	CGA, EGA
7	алфавитно-цифровой, ч/б	80x25	монохромный

8	графический, 16 цветов	160x200	PCjr
9	графический, 16 цветов	320x200	PCjr
10	графический, 4 цвета-PCjr 16 цветов-EGA	640x200	PCjr, EGA
13	графический, 16 цветов	320x200	EGA
14	графический, 16 цветов	640x200	EGA
15	графический, 4 цвета	640x350	EGA

А теперь приведем текст функции mode().

```
/* Установка видеорежима */
void mode(mode_code)
int mode_code;
     {
     union REGS r;
     r.h.al = mode_code;
     r.h.ah = 0;
     int86(0x10,&r,&r);
     }
```

В 4-ом режиме доступны две палитры (набора цветов). Каждая палитра определяет четыре цвета, отображаемые на экране терминала. В IBM PC палитра 0 определяет желтый, зеленый и красный цвета, палитра 1 определяет белый, ярко-красный (пурпурный) и голубой цвета. Для каждой палитры четвертым цветом является цвет фона, который обычно черный. BIOS-прерывание 16, функция 11, устанавливает палитру.

Функция pallet(), приведенная ниже, устанавливает номер палитры, который задается в качестве значения ее аргумента:

```
/* Установка палитры */
void palette(pnum)
int pnum;
     {
     union REGS r;
     r.h.bh=1;    /* код 4 графического режима */
     r.h.bl=pnum; /* номер палитры */
     r.h.ah=11;   /* устанавливается для вызова палитры */
     int86(0x10,&r,&r);
     }
```

ЗАПИСЬ ТОЧКИ РАСТРА

Одной из основных программ графики является программа записи точки растра - наименьшей адресуемой точки на экране дисплея. В данном случае термин "точка растра" будет использоваться для описания наименьшей адресуемой точки в отдельных графических режимах. Так как функция записи точки растра используется в других программах более высокого уровня, ее эффективность очень важна для быстродействия программ верхних уровней, которые непосредственно реализуют функции графики. На IBM PC и совместимых с ними компьютерах существуют два способа вывода информации с использованием точек растра. Первый способ, использующий прерывания ROM-BIOS, является наиболее простым, но и наименее быстродействующим (слишком медленным для наших целей). Вторым и более быстродействующими способом является размещение информации непосредственно в видеопамяти дисплея (ROM). Именно этот метод и рассматривается ниже.

Работа адаптеров CGA/EGA в графическеом режиме

Адаптер CGA всегда располагается в видеопамяти по адресу 8000000h. Адаптер EGA имеет аналогичное расположение для тех режимов, которые совместимы с режимами CGA (более полную информацию о аппаратных средствах поддержки графики вы можете получить в руководстве "IBM Technical Reference"). В 4

графическом режиме каждый байт содержит информацию о цвете для 4 точек растра (для каждой точки растра по 2 бита). Следовательно, для работы с экраном размерностью 320 на 200 требуется 16К памяти. Так как два бита могут содержать только 4 различных значения, в 4 видеорежиме поддерживаются только 4 цвета. Значение каждого двухбитового блока определяет цвет в соответствии с таблицей, приведенной ниже:

Значение	Цвет в палитре 0	Цвет в палитре 1
	фон	фон
1	желтый	голубой
2	красный	пурпурный
3	зеленый	булый

Особенность адаптера CGA заключается в том, что четные точки растра будут располагаться по адресу B8000000h, а нечетные - на 2000h (8152 - в десятичном виде) байтов выше, т.е. по адресу B8002000h. Следовательно, каждая строка точек растра требует 80 байтов (40 для четных точек растра, 40 - для нечетных). Внутри каждого байта точки располагаются слева направо, как они появляются на экране терминала. Это означает, что точка растра с номером 0 занимает 6 и 7 биты, в то время, как точка растра с номером 3 - 0 и 1 биты.

Так как каждый байт кодирует значение четырех точек растра, вы должны сохранять значение трех точек при изменении одной из них. Лучший способ сделать это - создание битовой маски со всеми битами, установленными в 1, кроме тех, которые будут изменяться. Значение битовой маски складывается по схеме "И" с действительным значением байта, а затем полученное значение складывается по схеме "ИЛИ" с новым значением. Однако ситуация несколько меняется, если вы хотите сложить по схеме "НЕ-ИЛИ" новое и старое значения. В этом случае вы просто складываете по схеме "ИЛИ" старое значение байта с 0, а затем складываете по схеме "НЕ-ИЛИ" новое двоичное представление цвета и получаете результат. Адрес требуемого байта определяется путем умножения значения координаты X на 40, а затем добавляется значение координаты Y, деленное на 4. Для того,чтобы определить, находится ли точка растра в четном или нечетном блоке памяти, используется остаток от деления значения координаты X на 2. Если остаток равен 0, блок является четным (используется первый блок), в противном случае - блок нечетный (используется второй блок). Требуемые биты внутри байта вычисляются путем выполнения деления по модулю 4. Остаток

определяет номер двухбитового пакета, который содержит информацию о требуемых точках растра. Для установки байта режима цвета и битовой маски используются операторы побитового сдвига. Хотя манипулирование битами в функции mempoint() несколько запутано, вы, однако, без труда разберетесь в ней, если тщательно изучите что именно и как она делает.

```
/* Запись точки в CGA/EGA */
     void mempoint(x,y,color_code)
     int x,y,color_code;
       {
       union mask {
          char c[2];
          int i;
       } bit_mask;
       int i,index,bit_position;
       unsigned char t;
       char xor; /* "НЕ-ИЛИ" цвета в случае его изменения */
       char far *ptr=(char far *) 0xB8000000; /* точка в памяти
                                                        CGA */
       bit_mask.i=0xFF3F; /* 11111111 00111111 в двоичном коде */
```

```
if(x<0 || x>199 || y<0 || y>319) return;
xor=color_code & 128; /* проверка, устанавливался ли
                         режим "НЕ-ИЛИ" */
color_code=color_code & 127; /* маска старших битов */
/*  установка битовой маски и битов режима цвета
    в правую позицию */
 bit_position=y%4; /* вычисление нужной позиции
                      в байте */
 color_code<<=2*(3-bit_position); /* сдвиг кода цвета
                                  в нужную позицию */
 bit_mask.i>>=2*bit_position; /* сдвиг битовой маски в
                                 нужную позицию */
 /* определение требуемого байта в памяти терминала */
  index=x*40+(y%4);
  if(x%2) index+=8152; /* если нечетный, используется
                          второй блок */
  /* запись цвета */
   if(!xor) {  /* режим изменения цвета */
      t=*(ptr+index) & bit_mask.c[0];
      *(ptr+index)=t|color_code;
   }
   else {
      t=*(ptr+index) | (char)0;
      *(ptr+index)=t & color_code;
   }
}
```

Заметим, что тип указателя видеопамяти объявлен как **far**; это необходимо, если вы транслируете в модели маленькой (small) памяти. Вы так же должны заметить, что специальный маркер режима записи XOR, определенный в функциях ROM-BIOS, используется и в функции mempoint().

ВЫЧЕРЧИВАНИЕ ЛИНИЙ

Функции вычерчивания линий являются основными подпрограммами графики и используются для отображения линий в заданном цвете путем задания ее начальных и конечных координат. В то время, как изображение вертикальных и горизонтальных линий не представляет особого труда, более трудной задачей является создание функций, которые рисуют линии вдоль диагоналей. Например, какие точки составляют линию, вычерчиваемую от точки с координатами 0,0 до точки с координатами 80,120?

Один из подходов к разработке функций вычерчивания линий использует отношение между смещением по координатам X и Y. Чтобы показать этот подход в действии, проведем линию между точками с координатами 0,0 и 5,10. Смещение по X равно 5, а по Y - 10. Отношение равно 1/2. Оно будет использоватся при определении коэффициента зависимости, по которому должны меняться координаты X и Y при изображении линий. В данном случае это означает, что приращение координаты X составляет половину величины изменения координаты Y. Начинающий программист часто использует этот метод при разработке функций вычерчивания линий. Хотя такой подход математически верен и прост для понимания, для его правильной работы и для того, чтобы избежать серьезных ошибок округления, необходимо использовать математические операции с числами с плавающей точкой. Это означает, что функции вычерчивания линий будут работать довольно медленно, если в систему не будет включен математический сопроцессор (например - Intel 8087). По этой причине этот метод используется довольно редко.

Наиболее общий метод изображения линий включает использование алгоритма Брезенхама. Хотя основой в нем служит также отношение между расстояниями по координатам X и Y, в данном случае не требуется выполнять деление или вычисление чисел с

плавающей точкой. Вместо этого, отношение между значениями координат X и Y представляется косвенным образом через серии сложений и вычитаний. Основной идеей алгоритма Брезенхама, является регистрация средних значений погрешностей между идеальным положением каждой точки и той позицией на экране дисплея, в которой она действительно отображается. Погрешность между идеальным и действительным положением точки возникает ввиду ограниченных возможностей технических средств. Фактически не существует дисплеев с бесконечно большой разрешающей способностью, и, следовательно, действительное положение каждой точки на линии требует наилучшей аппроксимации. В каждой итерации цикла вычерчивания линии вызываются две переменные xerr и yerr, которые увеличиваются в зависимости от изменения величин координат X и Y соответственно. Когда значение погрешности достигает определенного значения, оно вновь устанавливается в исходное положение, а соответствующий счетчик координат увеличивается. Этот процесс продолжается до тех пор, пока линия не будет полностью вычерчена. Функция line(), приведенная ниже, реализует этот метод. Вы должны изучать ее до тех пор, пока не поймете механизма выполнения всех ее операций. Заметим, что в ней

используется функция mempoint(), разработанная ранее для отображения точки на экране терминала.

```
 /* Вычерчивание линии заданного цвета с использованием
    алгоритма Брезенхама */
  void line(startx,starty,endx,endy,color)
  int startx,starty,endx,endy,color;
  {
    register int t,distance;
    int xerr=0,yerr=0,delta_x,delta_y;
    int incx,incy;
  /* вычисление расстояния в обоих направлениях  */
    delta_x=endx-startx;
    delta_y=endy-starty;
  /* определение направления шага,
     шаг вычисляется либо по вертикальной, либо горизонтальной
     линии   */
     if(delta_x>0) incx=1;
     else  if(delta_x==0) incx=0;
     else  incx= -1;
     if(delta_y>0) incy=1;
     else  if(delta_y==0) incy=0;
     else  incy= -1;
   /* определение какое расстояние больше */
     delta_x=abs(delta_x);
     delta_y=abs(delta_y);
     if(delta_x>delta_y) distance=delta_x;
     else distance=delta_y;
   /* вычерчивание линии */
     for (t=0; t<=distance+1; t++) {
        mempoint(startx,starty,color);
        xerr+=delta_x;
        yerr+=delta_y;
        if(xerr>distance) {
           xerr-=distance;
           startx+=incx;
        }
        if(yerr>distance) {
           yerr-=distance;
           starty+=incy;
        }
     }
  }
```

ИЗОБРАЖЕНИЕ И ЗАКРАШИВАНИЕ ПРЯМОУГОЛЬНИКОВ

Если у вас есть функции вычерчивания линий, то не составит особого труда создать функции вычерчивания прямоугольников. Пример, приведенный здесь, вычерчивает прямоугольники в заданном цвете путем задания координат двух противоположных углов.

```
/*  Вычерчивание прямоугольника  */
void box(startx,starty,endx,endy,color_code)
int startx,starty,endx,endy,color_code;
{
   line(startx,starty,endx,starty,color_code);
   line(startx,starty,startx,endy,color_code);
   line(startx,endy,endx,endy,color_code);
   line(endx,starty,endx,endy,color_code);
}
```

Для того, чтобы закрасить прямоугольник, требуется выполнить запись в каждую точку растра внутри прямоугольника. Программа fill_box(), приведенная ниже, закрашивает прямоугольник, определенный координатами двух противоположных углов, заданным цветом. В ней используется функция line(), задающая цвет внутри прямоугольника.

```
/* Закрашивание прямоугольника в заданный цвет */
void fill_box(startx,starty,endx,endy,color_code)
int startx,starty,endx,endy,color_code;
{
   register int i,begin,end;
   begin=startx<endx ? startx:endx;
   end=startx>endx ? startx:endx;
   for (i=begin;i<=end;++i)
      line(i,starty,i,endy,color_code);
 }
```

ВЫЧЕРЧИВАНИЕ ОКРУЖНОСТЕЙ

Самым быстрым и легким способом вычерчивания окружностей является способ, основанный на использовании опять таки алгоритма Брезенхама, похожего на одноименный алгоритм вычерчивания линий. Данный метод также не требует вычислений чисел с плавающей точкой, кроме вычисления коэффициента сжатия, поэтому он обеспечивает достаточное быстродействие. По существу, алгоритм основан на приращении координат X и Y на величину погрешности между ними. Значение погрешности содержится в переменой delta. Функция plot_circle() выполняет запись точек по окружности. Переменная asp_ratio является глобальной, т.к. она используется как в функции circle(), так и в функции plot_circle(). Эта переменная может быть полезна с точки зрения возможности установления ее значения вне функции circle() и дальнейшего использования внутри функции. Путем изменения значения этой переменной вы можете рисовать эллипсы. Параметрами функции circle() является центр окружности, ее радиус и цвет. Тексты функций приведены ниже.

```
 double asp_ratio;
/* Вычерчивание окружности с использованием алгоритма
   Брезенхама */
 void circle(x_center,y_center,radius,color_code)
 int x_center,y_center,radius,color_code;
 {
    register x,y,delta;
    asp_ratio=1.0; /* это число может меняется в различных
                      случаях */
     y=radius;
     delta=3-2*radius;
     for (x=0;x<y; ) {
         plot_circle(x,y,x_center,y_center,color_code);
```

```
            if(delta<0)
               delta+=4*x+6;
            else {
               delta+=4*(x-y)+10;
               y--;
            }
            x++;
         }
         x=y;
         if(y) plot_circle(x,y,x_center,y_center,color_code);
      }
    /* Функция печатает точки, определяющие окружность */
      void plot_circle(x,y,x_center,y_center,color_code)
      int x,y,x_center,y_center,color_code;
      {
         int startx,starty,endx,endy,x1,y1;
         starty=y*asp_ratio;

         endy=(y+1)*asp_ratio;
         startx=x*asp_ratio;
         endx=(x+1)*asp_ratio;
         for (x1=startx;x1<endx;++x1) {
            mempoint(x1+x_center,y+y_center,color_code);
            mempoint(x1+x_center,y_center-y,color_code);
            mempoint(x_center-x1,y+y_center,color_code);
            mempoint(x_center-x1,y_center-y,color_code);
         }
         for (y1=starty;y1<endy;++y1) {
            mempoint(y1+x_center,x+y_center,color_code);
            mempoint(y1+x_center,y_center-x,color_code);
            mempoint(x_center-y1,x+y_center,color_code);
            mempoint(x_center-y1,y_center-x,color_code);
         }
      }
```

Закрашивать окружность можно путем повторного вызова функции circle() с заданием все более и более меньшего радиуса. Этот способ используется в функции fill_circle(), текст которой приведен ниже.

```
    /* Закрашивание окружности путем повторного вызова
       circle() с уменьшением радиуса */
       void fill_circle(x,y,r,c)
       int x,y,r,c;
       {
          while (r) {
             circle(x,y,r,c);
             r--;
          }
        }
```

ПРОСТЕЙШАЯ ТЕСТОВАЯ ПРОГРАММЫ

Приведенные здесь программа иллюстрируют применение и возможности ранее описанных функций поддержки графики.

```
   /* Программа, иллюстрирующая работу графических
      функций */
    #include "dos.h"
    #include "stdio.h"
    void mode(),line(),box(),fill_box();
    void mempoint(),palette(),xhairs();
    void circle(),plot_circle(),fill_circle();
    double asp_ratio;
    main()
    {
    mode(4);
```

```
   palette(0);
   line(0,0,100,100,1);
   box(50,50,80,90,2);
   fill_box(100,0,120,40,3);
   circle(100,160,30,2);
   fill_circle(150,250,20,1);
   getchar();
   mode(2);
  }
       /* установка палитры */
   void palette(pnum)
   int pnum;
   {
      union REGS r;
      r.h.bh=1; /* код 4 режима графики */
      r.h.bl=pnum; /* номер палитры */
      r.h.ah=11; /* устанавливаетса для вызова палитры */
      int86(0x10,&r,&r);
   }
    /* Установка видеорежима */
      void mode(mode_code)
      int mode_code;
      {
          union REGS r;
          r.h.al = mode_code;
          r.h.ah = 0;
          int86(0x10,&r,&r);
       }
   /*  Вычерчивание прямоугольника  */
  void box(startx,starty,endx,endy,color_code)

  int startx,starty,endx,endy,color_code;
  {
     line(startx,starty,endx,starty,color_code);
     line(startx,starty,startx,endy,color_code);
     line(startx,endy,endx,endy,color_code);
     line(endx,starty,endx,endy,color_code);
  }
/* Вычерчивание линии заданного цвета с использованием
   алгоритма Брезенхама */
 void line(startx,starty,endx,endy,color)
 int startx,starty,endx,endy,color;
 {
   register int t,distance;
   int xerr=0,yerr=0,delta_x,delta_y;
   int incx,incy;
 /* Вычисление расстояния в обоих направлениях         */
   delta_x=endx-startx;
   delta_y=endy-starty;
 /* Определение направления шага,
    шаг вычисляется либо по вертикальной, либо по горизонтальной
    линии                    */
    if(delta_x>0) incx=1;
    else  if(delta_x==0) incx=0;
    else  incx= -1;
    if(delta_y>0) incy=1;
    else  if(delta_y==0) incy=0;
    else  incy= -1;
  /* Определение какое расстояние больше */
    delta_x=abs(delta_x);
    delta_y=abs(delta_y);
    if(delta_x>delta_y) distance=delta_x;
    else distance=delta_y;
  /* Вычерчивание линии */
```

```
   for (t=0; t<=distance+1; t++) {
      mempoint(startx,starty,color);
      xerr+=delta_x;
      yerr+=delta_y;
      if(xerr>distance) {
         xerr-=distance;
         startx+=incx;
      }
      if(yerr>distance) {
         yerr-=distance;
         starty+=incy;
      }
   }
}

 /* Закрашивание прямоугольника  заданным цветом */
 void fill_box(startx,starty,endx,endy,color_code)
 int startx,starty,endx,endy,color_code;
 {
    register int i,begin,end;
    begin=startx<endx ? startx:endx;
    end=startx>endx ? startx:endx;
    for (i=begin;i<=end;++i)
       line(i,starty,i,endy,color_code);
 }
 /* Вычерчивание окружности с использованием алгоритма
    Брезенхама */
  void circle(x_center,y_center,radius,color_code)
  int x_center,y_center,radius,color_code;
  {
     register x,y,delta;
     asp_ratio=1.0; /* это число может меняется в различных
                       случаях */
      y=radius;
      delta=3-2*radius;
      for (x=0;x<y; ) {
          plot_circle(x,y,x_center,y_center,color_code);
          if(delta<0)
             delta+=4*x+6;
          else {
             delta+=4*(x-y)+10;
             y--;
          }
          x++;
       }
       x=y;
       if(y) plot_circle(x,y,x_center,y_center,color_code);
    }
  /* Функция изображает точки, определяющие окружность */
    void plot_circle(x,y,x_center,y_center,color_code)
    int x,y,x_center,y_center,color_code;
    {
       int startx,starty,endx,endy,x1,y1;
       starty=y*asp_ratio;
       endy=(y+1)*asp_ratio;
       startx=x*asp_ratio;
       endx=(x+1)*asp_ratio;
       for (x1=startx;x1<endx;++x1) {
          mempoint(x1+x_center,y+y_center,color_code);
          mempoint(x1+x_center,y_center-y,color_code);
          mempoint(x_center-x1,y+y_center,color_code);
          mempoint(x_center-x1,y_center-y,color_code);

       }
```

```
        for (y1=starty;y1<endy;++y1) {
           mempoint(y1+x_center,x+y_center,color_code);
           mempoint(y1+x_center,y_center-x,color_code);
           mempoint(x_center-y1,x+y_center,color_code);
           mempoint(x_center-y1,y_center-x,color_code);
        }
     }
  /* Закрашивание окружности путем повторного вызова
       circle() с уменьшением радиуса */
       void fill_circle(x,y,r,c)
       int x,y,r,c;
       {
          while (r) {
             circle(x,y,r,c);
             r--;
          }
   }
   /* Запись точки в CGA/EGA */
    void mempoint(x,y,color_code)
    int x,y,color_code;
    {
       union mask {
          char c[2];
          int i;
       } bit_mask;
       int i,index,bit_position;
       unsigned char t;
       char xor; /* "исключающее ИЛИ" цвета в случае его
                    изменения */
       char far *ptr=(char far *) 0xB8000000; /* точка в памяти
                                                 CGA */
       bit_mask.i=0xFF3F;                 /* 11111111 00111111 в
                                             двоичном виде */
       if(x<0 || x>199 || y<0 || y>319) return;
       xor=color_code & 128;      /* проверка, устанавливался ли
                                     режим "исключающего ИЛИ" */
       color_code=color_code & 127;     /* маска старших битов */
       /*  Установка маски битов и битов режима цвета
           в правую позицию */
        bit_position=y%4; /* вычисление нужной позиции в байте */
        color_code<<=2*(3-bit_position); /* сдвиг кода цвета
                                    в нужную позицию */
        bit_mask.i>>=2*bit_position; /* сдвиг битовой маски в
                                    нужную позицию */
        /* определение требуемого байта в памяти терминала */

         index=x*40+(y%4);
         if(x%2) index+=8152; /* если нечетный, используется
                                 второй блок */
         /* Запись цвета */
          if(!xor) {  /* режим изменения цвета */
             t=*(ptr+index) & bit_mask.c[0];
             *(ptr+index)=t|color_code;
          }
          else {
             t=*(ptr+index) | (char)0;
             *(ptr+index)=t & color_code;
          }
       }
```

СОХРАНЕНИЕ И ЗАГРУЗКА ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

Сохранение и загрузка графических изображений является довольно простым делом, т.к. образ изображений находится в

видеопамяти дисплея, а ее содержимое легко копировать на дисковый файл. Главной проблемой является необходимость введения пользователем имени файла, ввиду того, что запрос о вводе и введенное имя файла сотрут часть информации на экране. Для того, чтобы избежать этого, разработаны функции save_pic() и load_pic(), тексты которых приводятся в данном разделе. Первая функция сохраняет 14 верхних строк изображения, чистит эту область, запрашивает имя файла и, после того, как оно будет введено, восстанавливает изображение.

```
   /* сохранение  графического изображения */
   void save_pic()
   {
     char fname[80];
     FILE *fp;
     register int i,j;
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                              памяти   CGA  */
     char far *temp;
     unsigned char buf[14][80]; /* содержит образ экрана */
     temp=ptr;
   /* сохранение верхних строк текущего содержимого экрана */
     for (i=0;i<14;++i)
        for (j=0;j<80;++j) {
           buf[i][j]=*temp; /* четный байт */
           buf[i][j+1]=*(temp+8152); /* нечетный байт */
           *temp=0; *(temp+8152)=0; /* чистка позиций экрана */
           temp++;
        }
      goto_xy(0,0);
      printf("Имя файла:");
      gets(fname);
      if(!(fp=fopen(fname,"wb"))) {
          printf("Фвайл не может быть открыт\n");
          return;
      }
      temp=ptr;
  /* восстановление содержимого экрана */
     for (i=0;i<14;++i)
        for (j=0;j<80;++j) {
           *temp= buf[i][j]; /* четный байт */
           *(temp+8125)=buf[i][j+1]; /* нечетный байт */
           *temp=0; *(temp+8152)=0; /* чистка позиций экрана */
           temp++;
        }
   /* копирование изображения в файл  */

      for (i=0;i<8152;i++) {
         putc(*ptr,fp); /* четный байт */
         putc(*(ptr+8125),fp); /* нечетный байт */
         ptr++;
      }
      fclose(fp);
   }
  /* загрузка изображения */
    void load_pic()
    {
       char fname[80];
       FILE *fp;
       register int i,j;
       char far *ptr=(char far *) 0xB8000000; /* точка в
                                                памяти   CGA  */
       char far *temp;
       unsigned char buf[14][80]; /* содержит образ экрана */
       temp=ptr;
```

```
   /* сохранение верхних строк текущего содержимого экрана */
       for (i=0;i<14;++i)
          for (j=0;j<80;j+=2) {
             buf[i][j]=*temp;
             buf[i][j+1]=*(temp+8152);
             *temp=0; *(temp+8152)=0; /* чистка позиций экрана */
             temp++;
          }
       goto_xy(0,0);
       printf("Имя файла:");
       gets(fname);
       if(!(fp=fopen(fname,"rb"))) {
          goto_xy(0,0);
          printf("Файл не может быть открыт\n");
          temp=ptr;
  /* восстановление содержимого экрана */
          for (i=0;i<14;++i)
             for (j=0;j<80;j+=2) {
                *temp= buf[i][j];
                *(temp+8125)=buf[i][j+1];
                temp++;
             }
          return;
        }
 /* загрузка изображения из файла */
      for (i=0;i<8152;i++) {
         *ptr=getc(fp); /* четный байт */
         *(ptr+8125)=getc(fp); /* нечетный байт */
         ptr++;
      }
      fclose(fp);

   }
```

Подпрограммы начинают обработку видеопамяти, начиная с адреса, содержащегося в указателе temp считывая или записывая каждый четный и нечетный байты в порядке возрастания их адресов. Такой подход позволяет добиться простоты и наглядности функционирования рассмотренных выше подпрограмм. В случае обработки видеопамяти в порядке возрастания адресов, вначале будет отображаться четная точка растра, а затем - нечетная.

ДУБЛИРОВАНИЕ ЧАСТИ ЭКРАНА

Иногда бывает полезным скопировать часть экрана в другую область. Это легко выполнить используя функцию copy(), текст которой приводится ниже.

```
     /* копирование части экрана в другую область */
     void copy(startx,starty,endx,endy,x,y)
     int startx,starty;      /* верхняя левая координата */
     int endx,endy;          /* нижняя правая координата области
                                копирования */
     int x,y;      /* верхняя левая координата области,
                      куда будет проводится копирование */
     {
        int i,j;
        unsigned char c;
        for (;startx<endx;startx++,x++)
           for (i=starty,j=y;i<endy;i++,j++) {
              c=read_point(startx,i); /* чтение точки */
              mempoint(x,j,c); /* запись ее в новую область */
            }
       }
```

Как вы могли убедиться, при обращении к функциим в качестве ее аргументов указываются верхняя левая и нижняя правая

координаты углов области, которая будет копироваться, и верхняя левая координаты, куда делается копия.

Вы также можете убедится, что с небольшими изменениями функцию copy() можно преобразовать в функцию move(). Функция move() пересылает указанную область в другую и чистит исходное место. Текст функции приводится ниже.

```
  /* Пересылка части экрана в другую область */
     void move(startx,starty,endx,endy,x,y)
     int startx,starty; /* верхняя левая координата */
     int endx,endy; /* нижняя правая координата области
                       пересылки */
     int x,y; /* верхняя левая координата области,
                 куда будет проводится пересылка */
     {
        int i,j;
        unsigned char c;
        for (;startx<endx;startx++,x++)
           for (i=starty;j=y;i<endy;i++,j++) {
              c=read_point(startx,i); /* чтение точки */
              mempoint(startx,i,0); /* стирание старого
                                       изображения */
              mempoint(x,j,c); /* запись точки в новую область */
            }
       }
```

ВРАЩЕНИЕ ТОЧКИ В ПЛОСКОСТИ ЭКРАНА

Вращение точки в плоскости экрана (двумерном пространстве) представляет собой довольно простую задачу в декартовой системе координат. Вы можете вспомнить из курса аналитической геометрии, что вращение точки вокруг центра на угол theta, описывается формулой:

```
new_x = old_x * cos(theta) - old_y * sin(theta)
new_y = old_x * sin(theta) - old_y * cos(theta)
```

Единственной сложностью при употреблении этих формул для графических дисплеев будет являтся тот факт, что экран дисплея не является декартовым пространством. Декартовы оси определяют 4 квадранта, как это показано на рисунке 4-2. Однако экран терминала представляет собой один из квадрантов, оси X и Y в котором перевернуты. Для решения этой проблемы необходимо определить новый центр и привести в соответствие координаты X и Y экрана и координаты осей декартова пространства. Любая точка на экране может быть использована в качестве центра, но обычно центр определяется как можно ближе к центру объекта, который мы собираемся вращать. Функция rotate_point(), приведенная ниже, вычисляет величину новых значений X и Y для заданного угла вращения.

```
   /* Вращение точки вокруг центра с координатами
      x_org и y_org на угол theta */
     void rotate_point(theta,x,y,x_org,y_org)
     double theta,*x,*y;
     int x_org,y_org;
     {
        double tx,ty;
   /* нормализация X и Y к начальному адресу */
        tx=*x-x_org;
        ty=*y-y_org;
   /* вращение */
         *x=tx*cos(theta)-ty*sin(theta);
         *y=tx*sin(theta)-ty*cos(theta);
    /* возвращение значений координат */
         *x+=x_org;
         *y+=y_org;
      }
```

Заметим, что rotate_point**()** изменяет параметры X и Y путем присвоения им требуемого значение для получения угла вращения, заданного переменной theta. Угол вращения задается в радианах.

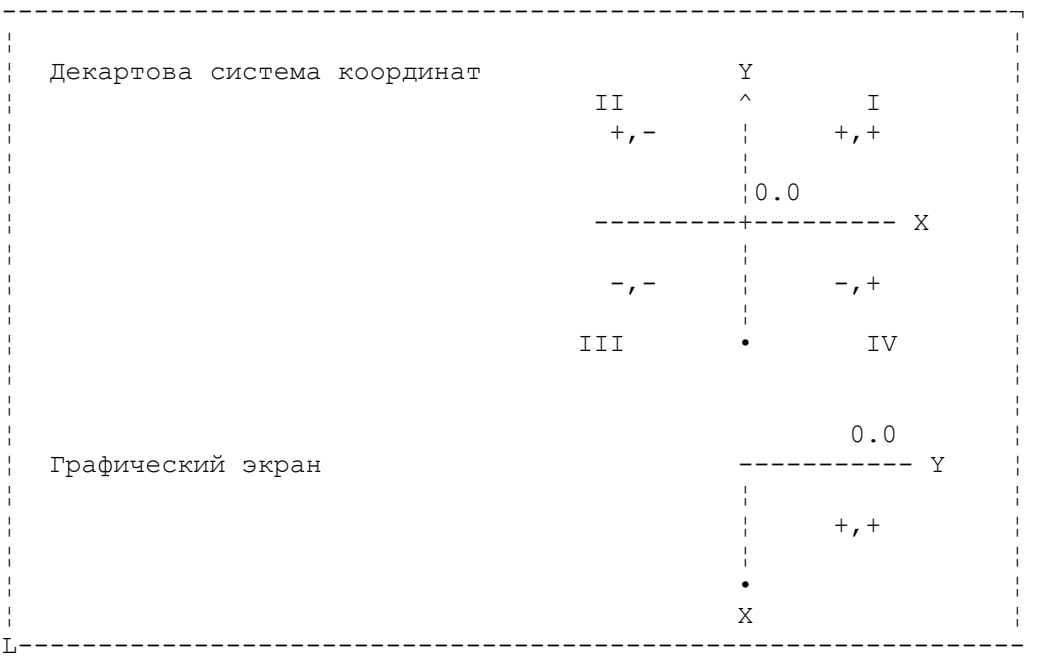


Рис. 4-2. Декартовы координаты на графическом экране.

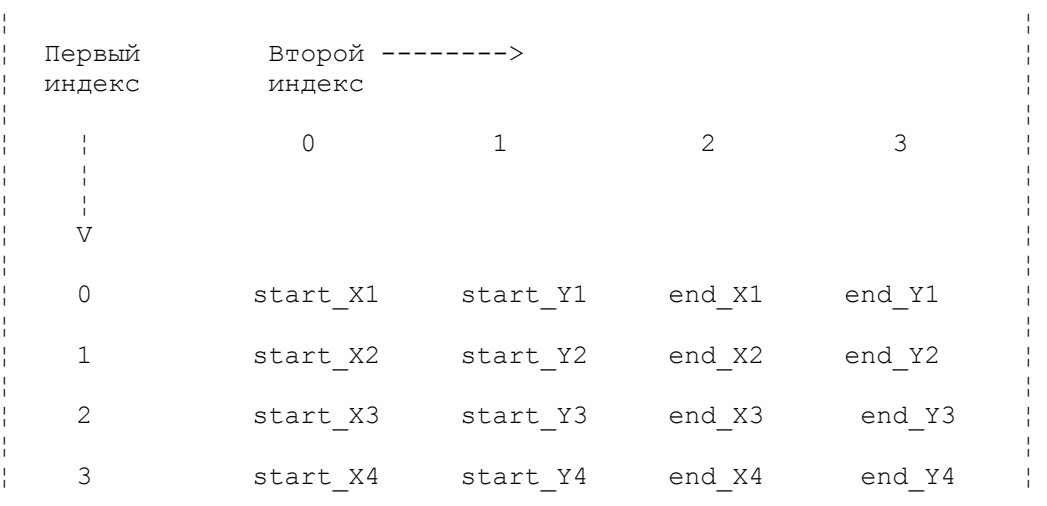
Вращение обьекта

Хотя функция rotate_point**()**, вычисляющая требуемые значения координат X и Y при вращении точки, уже была нами рассмотрена. Однако, она не может быть использована для вращения **(**поворотов**)** объектов. Для этого необходима другая функция. Под объектом здесь и далее будем понимать набор сегментов прямых отрезков. Координаты крайних точек каждого отрезка содержатся в двумерном массиве чисел с плавающей точкой. Каждая строка массива содержит начальные и конечные координаты данного отрезка. Это означает, что первая размерность массива представляет собой количество отрезков, входящих в состав объекта, а вторая размерность будет равна 4 **(**число координат крайних точек отрезка**)**. Например, массив, приведенный ниже

```
double object [10][4];
```

определяет объект, состоящий из 10 отрезков.

Как правило, массив организуется так, как показано на рисунке 4-3.



```
¦    .             .                                                ¦
¦    .             .                                                ¦
¦    .             .                                                ¦
¦    n          start_Xn     start_Yn     end_Xn      end_Yn      ¦
L-------------------------------------------------------------------
```

Рис. 4-3. Условная организация массива.

Определить объект - это значит разместить в массиве координаты начальных и конечных точек отрезков, составляющих объект. Например, если объект представляет собой прямоугольник вида:

то в массив, определяющий данный прямоугольник, заносятся следующие числа:

```
object[0][0] = 0; object[0][1] = 0;
object[0][0] = 0; object[0][3] = 10;
object[1][0] = 0; object[1][1] = 10;
object[1][0] = 10; object[1][3] = 10;
object[2][0] = 10; object[2][1] = 10;
object[2][0] = 10; object[2][3] = 0;
object[3][0] = 10; object[3][1] = 0;
object[3][0] = 0; object[3][3] = 0;
```

После того, как объект определен, вы можете вращать его, используя функцию rotate_object(), приведенную ниже, по часовой стрелке (клавиша <R>) или в противоположную сторону (клавиша <L>).

```
/* Вращение заданных объектов */
void rotate_object(ob, theta, x, y, sides)
double ob[][4]; /* описание объекта */
double theta;   /* угол поворота в радианах */
int x, y;
int sides;
  {
  register int i, j;
  double tempx, tempy;
  char ch;
  for(;;)
    {
    ch = getch(); /* ввод признака направления вращения */
    switch(tolower(ch))
      {
      case 'l': /* вращение против часовой стрелки */
        theta = theta < 0 ? -theta : theta;
        break;
      case 'r': /* вращение по часовой стрелке  */
        theta = theta > 0 ? -theta : theta;
        break;
      default: return;
      }
    for(j=0; j<=sides; j++) /* стирание старых линий  */

      {
      line((int) ob[j][0], (int) ob[j][1],
           (int) ob[j][2], (int) ob[j][3], 0);
      rotate_point(theta, &ob[j][0],
                   &ob[j][1], x, y);
      rotate_point(theta, &ob[j][2], &ob[j][3], x, y);
      line((int) ob[j][0], (int) ob[j][1],
           (int) ob[j][2], (int) ob[j][3], 2);
```

```
        }
      }
    }
```

Как показано в описании параметров функции rotate_object(), вращение осуществляется вокруг центра, заданного координатами X и Y, на угол, величина которого задана параметром theta в радианах. Минимальное значение параметра theta равно 0.01 радиан. Заметим, что объект сначала стирается из старой области размещения, а затем перерисовывается вновь. Если это условие не может быть выполнено, то экран окрашивается в голубой цвет. Необходимым условием выполнения программы rotate_object() является обязательное задание параметра sides.

Приведенная ниже функция display_object() не имеет отношения к вращению объектов, но она может быть полезна при работе с объектами. Она рисует на экране объекты, определенные в массиве ob.

```
/* отображение объекта на экране */
void display_object(ob, sides)
double ob[][4];
int sides;
{
  register int i;
  for(i=0; i<sides; i++)
    line((int)ob[i][0], (int)ob[i][1],
         (int)ob[i][2], (int)ob[i][3], 2);
  }
```

В качестве иллюстрации удобства использования функций вращения объектов ниже приводятся программы вращения изображения дома. На рисунке 4-4 показано изображение на экране терминала дома при различных углах поворота вокруг собственного центра. Прямоугольник, обрамляющий изображение вращаемого дома, поможет вам правильно оценить масштаб и перспективу.

Прим. пер. Рисунок 4-4 не может быть воспроизведен имеющимися средствами.

Рис. 4-4. Вращение объекта.

```
/* Пример вращения изображения объекта с использованием
   адаптера CGA/EGA в 4 графическом режиме
*/
#include "dos.h"
#include "stdio.h"
#include "math.h"
void mode(), line(), mempoint(), palette();
void rotate_point(), rotate_object(), display_object();
/* массив house определяет изображение дома */
double house[][4] =
  {
/*  startx,   starty,        endx,         endy  */
    120,         120,         120,          200,   /* дом */
    120,         200,          80,          200,
    80,          120,          80,          200,
    80,          120,         120,          120,
    60,          160,          80,          120,   /* крыша*/
    60,          160,          80,          200,
    120,         155,         100,          155,   /* двери*/
    100,         155,         100,          165,
    100,         165,         120,          165,
    90,          130,         100,          130,   /* окна */
    90,          130,          90,          140,
    100,         130,         100,          140,
    90,          140,         100,          140,
    90,          180,         100,          180,
```

```
 90,          180,          90,          190,
 100,         180,         100,          190
 };
main()
  {
  union k
    {
    char c[2];
    int i;
    }  key;
  mode(4);    /* режим   = 4 */
  palette(0); /* палитра = 0 */
  /* рисунок рамки,обрамляющей дом */
  line (30, 70, 30, 260, 2);
  line (160, 70, 160, 260, 2);
  line (30, 70, 160, 70, 2);
  line (30, 260, 160, 260, 2);
  display_object(house, 17);
  getchar();
  rotate_object(house, 0.025, 90, 160, 17);
  mode(3);

  }
/* Выбор палитры */
void palette(pnum)
int pnum;
  {
  union REGS r;
  r.h.bh = 1; /* код 4 графического режима */
  r.h.bl = pnum;
  r.h.ah = 11;
  int86(0x10, &r, &r);
  }
/* Выбор режима */
void mode(mode_code)
int mode_code;
  {
  union REGS r;
  r.h.al = mode_code;
  r.h.ah = 0;
  int86(0x10, &r, &r);
  }
/* Рисунок отрезка прямой заданного цвета */
void line(start_x, start_y, endx, endy, color)
int start_x, start_y, endx, endy, color;
  {
  register int t, distance;
  int x=0, y=0, delta_x, delta_y;
  int incx, incy;
  /* вычисление приращений по x и по y */
  delta_x = endx-start_x;
  delta_y = endy-start_y;
  /* вычисление признаков направлений отрезка */
  if(delta_x>0)
    incx=1;
  else
    if(delta_x==0)
      incx=0;
    else
      incx= -1;
  if(delta_y>0)
    incy=1;
  else
    if(delta_y==0)
```

```
        incy=0;
      else
        incy= -1;
    delta_x=abs(delta_x);
    delta_y=abs(delta_y);

    if(delta_x>delta_y)
      distance=delta_x;
    else
      distance=delta_y;
    /* рисунок отрезка */
    for(t=0; t<=distance; t++)
      {
      mempoint(start_x, start_y, color);
      x+=delta_x;
      y+=delta_y;
      if(x>distance)
        {
        x-=distance;
        start_x+=incx;
        }
      if(y>distance)
        {
        y-=distance;
        start_y+=incy;
        }
      }
    }
/* запись точки в CGA/EGA */
 void mempoint(x,y,color_code)
 int x,y,color_code;
 {
    union mask {
       char c[2];
       int i;
    } bit_mask;
    int i,index,bit_position;
    unsigned char t;
    char xor; /* "НЕ-ИЛИ" цвета в случае его
                 изменения */
    char far *ptr=(char far *) 0xB8000000; /* точка в
                                        памяти CGA */
    bit_mask.i=0xFF3F; /* 11111111 00111111 в
                          двоичном виде */
    if(x<0 || x>199 || y<0 || y>319) return;
    xor=color_code & 128; /* проверка, устанавливался ли
                             режим "НЕ-ИЛИ" */
    color_code=color_code & 127; /* маска старших битов */
    /*  установка битовой маски и битов режима цвета
        в правую позицию */
     bit_position=y%4; /* вычисление нужной позиции
                          в байте */
     color_code<<=2*(3-bit_position); /* сдвиг кода цвета
                                    в нужную позицию */
     bit_mask.i>>=2*bit_position; /* сдвиг битовой маски в

                                    нужную позицию */
     /* определение требуемого байта в памяти терминала */
      index=x*40+(y%4);
      if(x%2) index+=8152; /* если нечетный, используется
                              второй блок */
      /* запись цвета */
       if(!xor) {  /* режим изменения цвета */
          t=*(ptr+index) & bit_mask.c[0];
```

```
         *(ptr+index)=t|color_code;
       }
       else {
          t=*(ptr+index) | (char)0;
          *(ptr+index)=t & color_code;
       }
    }
/* вращение точки вокруг центра с координатами
   в x_org и y_org, на угол theta */
  void rotate_point(theta,x,y,x_org,y_org)
  double theta,*x,*y;
  int x_org,y_org;
  {
     double tx,ty;
/* нормализация X и Y к начальному адресу */
     tx=*x-x_org;
     ty=*y-y_org;
/* вращение */
      *x=tx*cos(theta)-ty*sin(theta);
      *y=tx*sin(theta)-ty*cos(theta);
 /* возвращение значений координат */
      *x+=x_org;
      *y+=y_org;
   }
  /* Вращение заданных объектов */
  void rotate_object(ob, theta, x, y, sides)
  double ob[][4]; /* описание объекта */
  double theta;   /* угол поворота в радианах */
  int x, y;
  int sides;
    {
    register int i, j;
    double tempx, tempy;

    char ch;
    for(;;)
      {
      ch = getch(); /* ввод признака направления вращения */
      switch(tolower(ch))
        {
        case 'l': /* вращение против часовой стрелки */
          theta = theta < 0 ? -theta : theta;
          break;
        case 'r': /* вращение по часовой стрелке  */
          theta = theta > 0 ? -theta : theta;
          break;
        default: return;
        }
      for(j=0; j<=sides; j++) /* стирание старых линий  */
        {
        line((int) ob[j][0], (int) ob[j][1],
             (int) ob[j][2], (int) ob[j][3], 0);
        rotate_point(theta, &ob[j][0],
                     &ob[j][1], x, y);
        rotate_point(theta, &ob[j][2], &ob[j][3], x, y);
        line((int) ob[j][0], (int) ob[j][1],
             (int) ob[j][2], (int) ob[j][3], 2);
        }
      }
    }
  /* отображение объекта на экране */
  void display_object(ob, sides)
  double ob[][4];
  int sides;
```

```
      {
      register int i;
      for(i=0; i<sides; i++)
        line((int) ob[i][0], (int) ob[i][1],
             (int) ob[i][2], (int) ob[i][3], 2);
      }
```

Сборка подпрограмм

В этом, последнем, параграфе, описывается простая программа рисования, использующая подпрограммы графики. Программы рисования часто используют "мышь", позволяющую пользователю удобным способом отображать линии на экране терминала. Однако, "мышью" комплектуются пока не все компьютеры поэтому, описанная здесь "программа-художник" ориентирована на операции с клавишами перемещения курсора.

В "рисующих" программах вам необходимо контролировать текущее положение координат X и Y (в графическом режиме текущие координаты индицируются курсором не совсем обычной формы). Для простоты дальнейшего изложения материала будем называть "графический" курсор "перекрестьем", принимая во внимание и тот факт, что в графическом режиме его форма действительно напоминает крест. Функция xhairs() размещает графический курсор в позиции, специфицированной значениями ее аргументов X и Y.

Напоминаем, что двоичный код цвета складывается по схеме "ИЛИ" со 128 с целью установки 7 бита в 1. Это позволяет функции mempoint() складывать по схеме "исключающего ИЛИ" двоичные коды старого и нового цвета на экране вместо его изменения. Такая возможность позволяет достичь двух важных моментов. Во-первых, графический курсор всегда видим, т.к. всегда имеет цвет, отличный от окружающего. Во-вторых, значительно упрощается процесс возврата точки растра, занимаемой курсором, в исходное положение. Эта операция выполняется путем повторного обращения к этим точкам (напомним, что последовательное выполнение двух операций по схеме "исключающего ИЛИ" всегда приводит к первоначальному значению). Ниже приведен текст функции отображения графического курсора.

```
/* отображение графического курсора */
 void xhairs(x,y)
 int x,y;
 {
    line(x-4,y,x+3,y,1|128);
    line(x,y+4,x,y-3,1|128);
 }
```

Программа рисования, описанная в данном разделе, позволит вам:

- рисовать линии;
- рисовать прямоугольники;
- закрашивать прямоугольники;
- рисовать окружности;
- закрашивать окружности;
- выбирать цвет;
- выбирать палитру;
- устанавливать скорость изменения параметров;

- сохранять графические изображения;
- загружать графические изображения;
- вращать объекты вокруг любой точки;
- копировать и пересылать графические изображения.

Приведем ниже текст главной программы :

```
main()
  {
  union k{
    char c[2];
```

```
  int i;
   } key ;
int x=10, y=10; /* текущая позиция экрана */
int cc=2;   /* текущий цвет */
int on_flag=1; /* признак использования карандаша */
int pal_num=1; /* номер палитры */
/* конечная точка определения линий,
  прямоугольников, окружностей */
int startx=0, starty=0, endx=0, endy=0;
int first_point=1;
int inc=1; /* шаг пересылки */
int sides=0; /* количество сторон выбранного объекта */
int i;
mode(4); /* переключатель режима CGA/EGA */
palette(0); /* палитра 0 */
xhairs(x, y); /* указатель курсора */
 do
  {
  key.i = bioskey(0);
  xhairs(x, y); /* графический курсор */
  if(!key.c[0]) switch(key.c[1])
    {
    case 75: /* влево */
      if(on_flag) line(x, y, x, y-inc, cc);
      y -= inc;
      break;
    case 77: /* вправо */
      if(on_flag) line(x, y, x, y+inc, cc);
      y += inc;
      break;
    case 72: /* вверх */
        if(on_flag) line(x, y, x-inc, y, cc);
      x -= inc;
      break;
    case 80: /* вниз */
        if(on_flag) line(x, y, x+inc, y, cc);
      x += inc;
      break;
    case 71: /* вверх и влево */

      if(on_flag) line(x, y, x-inc, y-inc, cc);
      x -= inc;
      y -= inc;
      break;
    case 73: /* вверх и вправо */
      if(on_flag) line(x, y, x-inc, y+inc, cc);
      x -= inc;
      y += inc;
      break;
    case 79: /* вниз и влево */
      if(on_flag) line(x, y, x+inc, y-inc, cc);
      x += inc;
      y -= inc;
      break;
    case 81: /* вниз и вправо */
      if(on_flag) line(x, y, x+inc, y+inc, cc);
      x += inc;
      y += inc;
      break;
    case 59: /* F1 - медленно */             inc=1;
      break;
    case 60: /* F2 - быстро   */
      inc=5;
      break;
```

```
          }
        else switch(tolower(key.c[0]))
          {
          case 'o': /* переключение шаблона */
            on_flag = !on_flag;
            break;
          case '1': cc=1; /* цвет 1 */
            break;
          case '2': cc=2; /* цвет 2 */
            break;
           case '3': cc=3; /* цвет 3 */
            break;
          case '0': cc=0; /* цвет 0 */
            break;
          case 'b': box(startx, starty, endx, endy, cc);
              break;
          case 'f':
            fill_box(startx, starty, endx, endy, cc); break;
          case 'l':
            line(startx, starty, endx, endy, cc);     break;
          case 'c':
            circle(startx, starty, endy-starty, cc);  break;
          case 'h':
            fill_circle(startx, starty, endy-starty, cc); break;
          case 's':
            save_pic();                               break;
          case 'r':
            load_pic();                               break;
          case 'm': /* пересылка фрагмента */
            move(startx, starty, endx, endy, x, y);   break;

          case 'x': /* копирование фрагмента */
            copy(startx, starty, endx, endy, x, y);   break;
          case 'd': /* определить поворот(сдвиг) объекта */
/* Внимание!! Во время трансляции программы идентификатор object
    был помечен как "неопределенный". Его описание действительно
    отсутствует в этой программе. (Ред. пер. И.Бычковский)
*/
            sides = define_objekt(object, x, y);      break;
          case 'a': /* поворот(сдвиг) объекта */
            rotate_objekt(object, 0.05, x, y, sides); break;
          case '\r': /* набор конечных точек для линий, кругов
                        или прямоугольников */
            if(first_point)
              { startx = x, starty = y; }
            else
              { endx = x, endy = y; }
            first_point = !first_point;break;
          case 'p':
            pal_num = pal_num==1 ? 2:1;
            palette(pal_num);
          }
        xhairs(x, y);
        }
      while (key.c[0]!='q');
      getchar();
      mode(2);
      }
```

Опишем кратко алгоритм работы программы рисования. Вначале экран терминала устанавливается в 4 графический режим. Затем устанавливается палитра 0, и графический курсор перемещается в верхний левый угол. Шаблон цвета по умолчанию устанавливается в соответствии с кодом 2 (красный в палитре 0). При перемещении графического курсора на экране остается след, который

окрашивается в соответствии с текущим цветом шаблона. Если нажимать клавиши перемещения курсора, графический курсор перемещается на одну точку растра в заданном направлении. Такая скорость перемещения может не удовлетворять пользователя, поэтому в программе предусмотрена возможность смещения на 5 точек растра путем нажатия клавиши F2. Отменить режим ускоренного перемещения можно путем нажатия клавиши F1. Изменение цвета осуществляется при нажатии цифровых клавиш от 0 до 3. В палитре 0 цифра 0 зарезервирована, 1 определяет зеленый цвет, 2 - красный, 3 - желтый. Шаблон цвета может быть изменен путем нажатия клавиши 0. Клавиши <Курсор в левый верхний угол> (<HOME>), <Страница вверх> (<PGUP>), <Страница вниз> (<PGDN>) и <Кон> (<END>) перемещают графический курсор в указанном направлении и под углом в 45 градусов.

Для анализа кодов операций чтения в программах используется функция bioskey(). Порядок подключения этой функции к программе

при компиляции описан в главе 1. В программу включены обращения к функциям, позволяющим вам рисовать и закрашивать прямоугольники и окружности, рисовать линии, копировать и перемещать изображение на экране, сохранять на диске и загружать с него содержимое экрана, отображать и вращать объекты.

При изображении линий, прямоугольников и окружностей вам необходимо определить координаты двух точек. Для прямоугольников - это координаты двух противоположных углов. Для линий задается начальная и конечная точки, а для окружности - координаты центра и точки, через которую она будет проходить.

Процесс выбора этих точек выполняется путем нажатия клавиши <ВВОД> в момент, когда графический курсор находится в требуемой области. Например, для изображения линии вы перемещаете графический курсор в точку, где она должна начинаться и нажимаете клавишу <ВВОД> . Затем вы устанавливаете курсор в точку, где линия заканчивается, и нажимаете <ВВОД> снова. При нажатии клавиши <ВВОД> выполняется загрузка переменных startx, starty, endx и endy, которые потом используются в качестве параметров вызываемых функций. После того, как координаты точек будут определены, при нажатии клавиши <B> рисуется квадрат, а <F> - квадрат закрашивается, при нажатии <L> рисуется линия, при нажатии <C> рисуется окружность, а <H> - окружность закрашивается.

Для копирования или перемещения части экрана вы должны определить верхний левый и нижний правый углы области, которую вы хотите переместить (нажатием клавиши <ВВОД>). Затем вы перемещаете курсор в верхний левый угол области, куда вы хотите переместить изображение. Для пересылки изображения требуется нажать клавишу <M>, а для копирования - <X>. Запомните, что старое изображение в области, куда осуществляется копирование, будет уничтожено.

Для вращения объекта вам необходимо определить сам объект, путем нажатия клавиши <D>. Затем, используя клавишу <ВВОД>, вы должны определить начальные и конечные координаты точек для отрезков по периметру выбранного объекта. Процесс выбора объекта вращения и определения его границ реализуется функцией define_object(). Вращение объекта начинается после нажатия клавиши <A>. Для определения направления вращения используются клавиши <L> (по часовой стрелке) или <R> (против часовой стрелки). Остановить процесс вращения можно нажатием любой клавиши, кроме <L> или <A>.

Для остановки работы программы используется клавиша <Q>. При желании вы можете включить в программу функции для работы с "мышью". Пример выходных данных программы показан на рисунке 4-5.

Рисунок 4-5 на стр. 163 не может быть воспроизведен имеющимися

средствами. (Ред. пер. И.Бычковский)

Рис. 4-5. Простейшие результаты работы программы рисования.

А теперь приведем всю программу рисования целиком.

```
/* Программа для CGA/EGA, позволяющая рисовать линии,
прямоугольники и окружности. Вы можете нарисовать какой-либо
объект и вращать его по часовой или против часовой стрелки. Вы
так же можете копировать графическое изображение на диск и
загружать его с диска. */
     #define NUM_SIDES 20 /* число сторон объекта;
                             при необходимости увеличивается */
     #include "dos.h"
     #include "stdio.h"
     #include "math.h"
     void mode(), line(), box(), fill_box();
     void mempoint(), palette(), xhairs();
     void circle(), plot_circle(), fill_circle();
     void rotate_point(), rotate_object(), goto_xy();
     void display_object(), copy(), move();
     void save_pic(), load_pic();
     unsigned char read_point();
     /*  Этот массив содержит динамически меняющиеся
         координаты объекта.
     */
     double object[NUM_SIDES][4];
     double asp_ratio; /* содержит коэффициент сжатия для
                          окружностей */
     main()
       {
       union k{
         char c[2];
         int i;
          } key ;
       int x=10, y=10; /* текущая позиция экрана */
       int cc=2;   /* текущий цвет */
       int on_flag=1; /* признак использования карандаша */
       int pal_num=1; /* номер палитры */
       int startx=0, starty=0, endx=0, endy=0;
       int first_point=1;
       int inc=1; /* шаг пересылки */
       int sides=0; /* количество сторон выбранного объекта */
       int i;
       mode(4); /* переключатель режима CGA/EGA */


       palette(0); /* палитра 0 */
       xhairs(x, y); /* указатель курсора */
        do
         {
         key.i = bioskey(0);
         xhairs(x, y);
         if(!key.c[0]) switch(key.c[1])
           {
           case 75: /* влево */
             if(on_flag) line(x, y, x, y-inc, cc);
             y -= inc;
             break;
           case 77: /* вправо */
             if(on_flag) line(x, y, x, y+inc, cc);
             y += inc;
             break;
           case 72: /* вверх */
```

```
      if(on_flag) line(x, y, x-inc, y, cc);
    x -= inc;
    break;
  case 80: /* вниз */
      if(on_flag) line(x, y, x+inc, y, cc);
    x += inc;
    break;
  case 71: /* вверх и влево */
    if(on_flag) line(x, y, x-inc, y-inc, cc);
    x -= inc;
    y -= inc;
    break;
  case 73: /* вверх и вправо */
    if(on_flag) line(x, y, x-inc, y+inc, cc);
    x -= inc;
    y += inc;
    break;
  case 79: /* вниз и влево */
    if(on_flag) line(x, y, x+inc, y-inc, cc);
    x += inc;
    y -= inc;
    break;
  case 81: /* вниз и вправо */
    if(on_flag) line(x, y, x+inc, y+inc, cc);
    x += inc;
    y += inc;
    break;
  case 59: /* F1 - медленно */
    inc=1;
    break;
  case 60: /* F2 - быстро   */
    inc=5;
    break;
  }
else switch(tolower(key.c[0]))
  {


  case 'o': /* переключение шаблона */
    on_flag = !on_flag;
    break;
  case '1': cc=1; /* цвет 1 */
    break;
  case '2': cc=2; /* цвет 2 */
    break;
   case '3': cc=3; /* цвет 3 */
    break;
  case '0': cc=0; /* цвет 0 */
    break;
  case 'b':
    box(startx, starty, endx, endy, cc);      break;
  case 'f':
    fill_box(startx, starty, endx, endy, cc); break;
  case 'l':
    line(startx, starty, endx, endy, cc);     break;
  case 'c':
    circle(startx, starty, endy-starty, cc);  break;
  case 'h':
    fill_circle(startx, starty, endy-starty, cc); break;
  case 's':
    save_pic();                               break;
  case 'r':
    load_pic();                               break;
  case 'm': /* пересылка фрагмента */
```

```
            move(startx, starty, endx, endy, x, y);   break;
          case 'x': /* копирование фрагмента */
            copy(startx, starty, endx, endy, x, y);   break;
          case 'd': /* определить объект вращения */
            sides = define_objekt(object, x, y);      break;
          case 'a': /* вращение объекта */
            rotate_objekt(object, 0.05, x, y, sides); break;
          case '\r': /* набор конечных точек для линий, кругов
                        или прямоугольников */
            if(first_point)
              { startx = x, starty = y; }
            else
              { endx = x, endy = y; }
            first_point = !first_point;               break;
          case 'p':
            pal_num = pal_num==1 ? 2:1;
            palette(pal_num);
          }
        xhairs(x, y);
        }
      while (key.c[0]!='q');
      getchar();
      mode(2);
      }
    /* установка палитры */
    void palette(pnum)
    int pnum;


      {
      union REGS r;
      r.h.bh = 1; /* код 4 режима графики */
      r.h.bl = pnum;
      r.h.ah = 11; /* установка палитры */
      int86(0x10, &r, &r);
      }
    /* установка видео-режима */
    void mode (mode_code)
    int mode_code;
      {
      union REGS r;
      r.h.al = mode_code;
      r.h.ah = 0;
      int86(0x10,&r, &r);
      }
    /* изображение прямоугольника */
    void box(sx, sy, ex, ey, c)
    int sx, sy, ex, ey, c;
      {
      line(sx, sy, ex, sy, c);
      line(sx, sy, sx, ey, c);
      line(sx, ey, ex, ey, c);
      line(ex, sy, ex, ey, c);
      }
/* изображение линии заданного цвета с использованием
   алгоритма Брезенхама */
 void line(startx,starty,endx,endy,color)
 int startx,starty,endx,endy,color;
 {
   register int t,distance;
   int xerr=0,yerr=0,delta_x,delta_y;
   int incx,incy;
 /* вычисление расстояния в обоих направлениях         */
   delta_x=endx-startx;
```

```
  delta_y=endy-starty;
/* определение направления шага,
   шаг вычисляется либо по вертикальной, либо горизонтальной
   линии                    */
   if(delta_x>0) incx=1;
   else  if(delta_x==0) incx=0;
   else  incx= -1;
   if(delta_y>0) incy=1;
   else  if(delta_y==0) incy=0;
   else  incy= -1;
 /* определение какое расстояние больше */


   delta_x=abs(delta_x);
   delta_y=abs(delta_y);
   if(delta_x>delta_y) distance=delta_x;
   else distance=delta_y;
 /* изображение линии */
   for (t=0; t<=distance+1; t++) {
      mempoint(startx,starty,color);
      xerr+=delta_x;
      yerr+=delta_y;
      if(xerr>distance) {
         xerr-=distance;
         startx+=incx;
      }
      if(yerr>distance) {
         yerr-=distance;
         starty+=incy;
      }
   }
}
 /* закрашивание прямоугольника в заданный цвет */
 void fill_box(startx,starty,endx,endy,color_code)
 int startx,starty,endx,endy,color_code;
 {
    register int i,begin,end;
    begin=startx<endx ? startx:endx;
    end=startx>endx ? startx:endx;
    for (i=begin;i<=end;++i)
       line(i,starty,i,endy,color_code);
 }
 /* изображение   окружности   с   использованием   алгоритма
    Брезенхама */
  void circle(x_center,y_center,radius,color_code)
  int x_center,y_center,radius,color_code;
  {
     register x,y,delta;
     asp_ratio=1.0; /* это число меняется в различных
                       случаях */
      y=radius;
      delta=3-2*radius;
      for (x=0;x<y; ) {
          plot_circle(x,y,x_center,y_center,color_code);
          if(delta<0)
             delta+=4*x+6;
          else {
             delta+=4*(x-y)+10;


             y--;
          }
          x++;
       }
```

```
      x=y;
      if(y) plot_circle(x,y,x_center,y_center,color_code);
    }

  /* plot_circle печатает точки, определяющие окружность */
    void plot_circle(x, y, x_center, y_center, color_code)
    int x_center,y_center,radius,color_code;
    {
       int x, y, startx, starty, endx, endy, x1, y1;
       starty=y*asp_ratio;
       endy=(y+1)*asp_ratio;
       startx=x*asp_ratio;
       endx=(x+1)*asp_ratio;
       for (x1=startx;x1<endx;++x1) {
          mempoint(x1+x_center,y+y_center,color_code);
          mempoint(x1+x_center,y_center-y,color_code);
          mempoint(x_center-x1,y+y_center,color_code);
          mempoint(x_center-x1,y_center-y,color_code);
       }
       for (y1=starty;y1<endy;++y1) {
          mempoint(y1+x_center,x+y_center,color_code);
          mempoint(y1+x_center,y_center-x,color_code);
          mempoint(x_center-y1,x+y_center,color_code);
          mempoint(x_center-y1,y_center-x,color_code);
       }
    }
}
/* закрашивание окружности путем повторного вызова
     circle() с уменьшением радиуса */
     void fill_circle(x,y,r,c)
     int x,y,r,c;
     {
        while (r) {
           circle(x,y,r,c);
           r--;
        }
     }
 /* запись точки в CGA/EGA */
  void mempoint(x,y,color_code)
  int x,y,color_code;
  {
     union mask {
        char c[2];
        int i;
     } bit_mask;


     int i,index,bit_position;
     unsigned char t;
     char xor; /* "исключающее ИЛИ" цвета в случае его
                  изменения */
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                              памяти CGA */
     bit_mask.i=0xFF3F; /* 11111111 00111111 в
                           двоичном виде */
     if(x<0 || x>199 || y<0 || y>319) return;
     xor=color_code & 128; /* проверка, устанавливался ли
                              режим "исключающего ИЛИ" */
     color_code=color_code & 127; /* маска старших битов */
     /*  установка битовой маски и битов режима цвета
         в правую позицию */
      bit_position=y%4; /* вычисление нужной позиции
                           в байте */
      color_code<<=2*(3-bit_position); /* сдвиг кода цвета
```

```
                                  в нужную позицию */
      bit_mask.i>>=2*bit_position; /* сдвиг битовой маски в
                                     нужную позицию */
      /* определение требуемого байта в памяти терминала */
       index=x*40+(y%4);
       if(x%2) index+=8152; /* если нечетный, используется
                               второй блок */
       /* запись цвета */
        if(!xor) {  /* режим изменения цвета */
           t=*(ptr+index) & bit_mask.c[0];
           *(ptr+index)=t|color_code;
        }
        else {
           t=*(ptr+index) | (char)0;
           *(ptr+index)=t & color_code;
        }
     }
 /* отображение графического курсора */
  void xhairs(x,y)
  int x,y;
  {
     line(x-4,y,x+3,y,1|128);
     line(x,y+4,x,y-3,1|128);
   }
 /* чтение байта из оперативной памяти CGA/EGA */
  unsigned char read_point(x,y)
  int x,y;
  {


     union mask {
        char c[2];
        int i;
     } bit_mask;
     int i,index,bit_position;
     unsigned char t;
     char xor; /* "исключающее ИЛИ" цвета в случае его
                  изменения */
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                               памяти CGA */
     bit_mask.i=3; /* 11111111 00111111 в
                      двоичном виде */
     if(x<0 || x>199 || y<0 || y>319) return 0;
     /*  установка битовой маски и битов режима цвета
         в правую позицию */
      bit_position=y%4; /* вычисление нужной позиции
                           в байте */
      bit_mask.i<<=2*(3-bit_position);
      /* определение требуемого байта в памяти терминала */
       index=x*40+(y>>4);
       if(x%2) index+=8152; /* если нечетный, используется
                               второй блок */
       /* запись цвета */
        t=*(ptr+index) & bit_mask.c[0];
        t>>=2*(3-bit_position);
        return t;
     }
/* сохранение  графического изображения  */
 void save_pic()
 {
   char fname[80];
   FILE *fp;
   register int i,j;
   char far *ptr=(char far *) 0xB8000000; /* точка в
```

```
                                                памяти   CGA  */
   char far *temp;
   unsigned char buf[14][80]; /* содержит образ экрана */
   temp=ptr;
 /* сохранение верхних строк текущего содержимого экрана */
   for (i=0;i<14;++i)
      for (j=0;j<80;++j) {
         buf[i][j]=*temp; /* четный байт */
         buf[i][j+1]=*(temp+8152); /* нечетный байт */
         *temp=0; *(temp+8152)=0; /* чистка позиций экрана */
         temp++;
      }
    goto_xy(0,0);
    printf("Имя файла:");


    gets(fname);
    if(!(fp=fopen(fname,"wb"))) {
        printf("Файл не может быть открыт\n");
        return;
    }
    temp=ptr;
/* восстановление содержимого экрана */
   for (i=0;i<14;++i)
      for (j=0;j<80;++j) {
         *temp= buf[i][j]; /* четный байт */
         *(temp+8125)=buf[i][j+1]; /* нечетный байт */
         *temp=0; *(temp+8152)=0; /* чистка позиций экрана */
         temp++;
      }
 /* копирование изображения в файл  */
    for (i=0;i<8152;i++) {
       putc(*ptr,fp); /* четный байт */
       putc(*(ptr+8125),fp); /* нечетный байт */
       ptr++;
    }
    fclose(fp);
 }
/* загрузка изображения */
  void load_pic()
  {
     char fname[80];
     FILE *fp;
     register int i,j;
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                               памяти   CGA  */
     char far *temp;
     unsigned char buf[14][80]; /* содержит образ экрана */
     temp=ptr;
 /* сохранение верхних строк текущего содержимого экрана */
     for (i=0;i<14;++i)
        for (j=0;j<80;j+=2) {
           buf[i][j]=*temp;
           buf[i][j+1]=*(temp+8152);
           *temp=0; *(temp+8152)=0; /* чистка позиций экрана */
           temp++;
        }
     goto_xy(0,0);
     printf("Имя файла:");
     gets(fname);
     if(!(fp=fopen(fname,"rb"))) {
        goto_xy(0,0);
        printf("Файл не может быть открыт\n");
```

```
        temp=ptr;
 /* восстановление содержимого экрана */
        for (i=0;i<14;++i)
           for (j=0;j<80;j+=2) {
              *temp= buf[i][j];
              *(temp+8125)=buf[i][j+1];
              temp++;
           }
        return;
      }
/* загрузка изображения из файла */
    for (i=0;i<8152;i++) {
       *ptr=getc(fp); /* четный байт */
       *(ptr+8125)=getc(fp); /* нечетный байт */
       ptr++;
    }
    fclose(fp);
 }
 /*  поместить курсор в заданное положение */
    void goto_xy(x,y)
    int x,y;
    {
       r.h.ah=2; /* адресация курсора */
       r.h.dl=y; /* координата столбца */
       r.h.dh=x; /* координата строки */
       r.h.bh=0; /* видеостраница */
       int86(0x10,&r,&r);
     }
 /* копирование части экрана в другую область */
   void copy(startx,starty,endx,endy,x,y)
   int startx,starty; /* верхняя левая координата */
   int endx,endy; /* нижняя правая координата области
                     копирования */
   int x,y; /* верхняя левая координата области,
               куда будет проводится копирование */
   {
      int i,j;
      unsigned char c;
      for (;startx<endx;startx++,x++)
         for (i=starty,j=y;i<endy;i++,j++) {
            c=read_point(startx,i); /* чтение точки */
            mempoint(x,j,c); /* запись ее в новую область */
          }
     }
 /* пересылка части экрана в другую область */


   void move(startx,starty,endx,endy,x,y)
   int startx,starty; /* верхняя левая координата */
   int endx,endy; /* нижняя правая координата области
                     пересылки */
   int x,y; /* верхняя левая координата области,
               куда будет проводиться пересылка */
   {
      int i,j;
      unsigned char c;
      for (;startx<endx;startx++,x++)
         for (i=starty,j=y;i<endy;i++,j++) {
            c=read_point(startx,i); /* чтение точки */
            mempoint(startx,i,0); /* стирание старого
                                     изображения */
            mempoint(x,j,c); /* запись точки в новую область */
          }
```

```
    }
/* вращение точки вокруг центра с координатами
   x_org и y_org на угол theta */
  void rotate_point(theta,x,y,x_org,y_org)
  double theta,*x,*y;
  int x_org,y_org;
  {
     double tx,ty;
/* нормализация X и Y к начальному адресу */
     tx=*x-x_org;
     ty=*y-y_org;
/* вращение */
      *x=tx*cos(theta)-ty*sin(theta);
      *y=tx*sin(theta)-ty*cos(theta);
 /* возвращение значений координат */
      *x+=x_org;
      *y+=y_org;
   }
  /* Вращение заданных объектов */
  void rotate_object(ob, theta, x, y, sides)
  double ob[][4]; /* описание объекта */
  double theta;   /* угол поворота в радианах */
  int x, y;
  int sides;
    {
    register int i, j;
    double tempx, tempy;
    char ch;


    for(;;)
      {
      ch = getch(); /* ввод признака направления вращения */
      switch(tolower(ch))
        {
        case 'l': /* вращение против часовой стрелки */
          theta = theta < 0 ? -theta : theta;
          break;
        case 'r': /* вращение по часовой стрелке  */
          theta = theta > 0 ? -theta : theta;
          break;
        default: return;
        }
      for(j=0; j<=sides; j++) /* стирание старых линий  */
        {
        line((int) ob[j][0], (int) ob[j][1],
             (int) ob[j][2], (int) ob[j][3], 0);
        rotate_point(theta, &ob[j][0], &ob[j][1], x, y);
        rotate_point(theta, &ob[j][2], &ob[j][3], x, y);
        line((int) ob[j][0], (int) ob[j][1],
             (int) ob[j][2], (int) ob[j][3], 2);
        }
      }
    }
  /* отображение объекта на экране */
  void display_object(ob, sides)
  double ob[][4];
  int sides;
    {
    register int i;
    for(i=0; i<sides; i++)
      line((int)ob[i][0], (int)ob[i][1],
           (int)ob[i][2], (int)ob[i][3], 2);
    }
```

```
/* определение объекта по заданным точкам */
     define_object(ob,x,y)
     double ob[][4];
     int x,y;
     {
     union k{
       char c[2];
       int i;
        } key ;
     register int i,j;
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                               памяти   CGA  */
     char far *temp;


     unsigned char buf[14][80]; /* содержит образ экрана */
     int sides=0;
     temp=ptr;
 /* сохранение верхних строк текущего содержимого экрана */
     for (i=0;i<14;++i)
        for (j=0;j<80;j+=2) {
           buf[i][j]=*temp;
           buf[i][j+1]=*(temp+8152);
           *temp=0; *(temp+8152)=0; /* чистка позиций экрана */
           temp++;
        }
       i=0;
       xhairs(x,y);
       do {
          goto_xy(0,0);
      printf("Определите сторону %d,",sides+1);
          if(i==0) printf("Введите первую габаритную точку ");
          else printf("Введите вторую габаритную точку ");
          key.i=bioskey(0);
          xhairs(x,y);
          if(key.c[0]=13) {
             ob[sides][++i]=(double) x;
             ob[sides][++i]=(double) y;
             if(i==4) {
                i=0;
                sides++;
             }
           }
          /* перемещение графического курсора */
          if(rey.c[0]) switch (key.c[1]) {
             case 75: /* влево */
                y-=1;
                break;
               case 77: /* вправо */
                y += 1
                break;
              case 72: /* вверх */
                     x -= 1;
                break;
              case 80: /* вниз */
                     x += 1;
                break;
              case 71: /* вверх и влево */
                   x -= 1;
                y -= 1;
                break;
              case 73: /* вверх и вправо */
                   x -= 1;
```

```
                y += 1;
                break;
              case 79: /* вниз и влево */
                x += 1;
                y -= 1;
                break;
              case 81: /* вниз и вправо */
                   x += 1;
                y += 1;
                break;
            }
            if(key.c[1]!=59) xhairs(x,y);
         }  while (key.c[1]!=59); /* нажата клавиша F1 */
         /* восстановление содержимого экрана */
         for (i=0;i<14;++i)
            for (j=0;j<80;j+=2) {
               *temp= buf[i][j];
               *(temp+8125)=buf[i][j+1];
               temp++;
            }
         return sides;
      }
```

Хотя в этой программе довольно много операторов, вам надо ввести ее в свой компьютер, так как она довольно интересна. Также она будет являться хорошим графическим инструментальным средством, которое вы сможете использовать в любой момент времени.

ВИДЕОИГРЫ

Видеоигры приносят вам либо радость, либо огорчения. Это зависит от вашего отношения к ним. Однако программирование видеоигр не только интересно, но и полезно. Фактически, одним из лучших способов обогащения является разработка удачных видеоигр. Хорошая игра, объединяющая логику с живой графикой, доставит большое удовольствие игроку. Лучшие из видеоигр, включающие элементы искусственного интеллекта, позволяют компьютеру вести диалог с игроком и "осмысленно" реагировать на ввод данных.

В этой главе вы ознакомитесь с некоторыми основами техники программирования видеоигр, что позволит вам разрабатывать собственные игры. Вы научитесь "оживлять" различные объекты на экране вашего терминала. Разработка видеоигр явится для вас отправной точкой. Многие принципы, используемые при разработке видеоигр, будут полезны для вас и увеличат ваш интерес к работе. Для использования программ, приводимых в качестве примера в этой главе, необходим компьютер IBM PC или другой, совместимый с ним, в состав которого входят адаптеры CGA, EGA или VGA. Многие из функций, используемых в данной главе, рассматривались в главе 4. Поэтому, если вы еще не изучили главу 4, то вам придется сделать это сейчас.

СПРАЙТЫ

Многие видеоигры, в которых игрок управляет объектами, атакующими другие объекты, управляемые программой или защищающимися от них, включают два класса активных объектов: среду (представляющую для нас маломеняющееся поле игры) и спрайты. СПРАЙТ - это небольшой подвижный объект, который движется по полю видеоигры по определенным правилам с заданной целью. Например, когда космический корабль стреляет фотонными торпедами, изображение торпеды реализуется спрайтом. В рамках данной главы под спрайтом будем понимать фигуру, определенную

некоторыми замкнутыми отрезками (многоугольник). Хотя, в общем случае спрайт может изображаться любым образом, например, в виде окружности. В примерах, рассматриваемых в данной главе, определять спрайт будем в виде двумерного массива целых чисел. Например, спрайт, состоящий из 4 отрезков может быть описан следующим массивом

```
int sprite [4][4];
```

Первая размерность массива определяет количество отрезков спрайта, а вторая - координаты конечных точек отрезков (подобный способ описания объектов подробно рассмотрен в главе 4). Начальные и конечные координаты отрезков задаются в следующей последовательности:

```
start_x, start_y, end_x, end_y
```

Отрезок, входящий в спрайт, с координатами конечных точек 0,0 и 0,10 может быть описан следующим массивом:

```
sprite[0][0] = 0;   /* start_x */
sprite[0][1] = 0;   /* start_y */
sprite[0][2] = 0;   /* end_x */
sprite[0][3] = 10;  /* end_y */
```

ПОЛЕ ИГРЫ

В большинстве видеоигр поле игры представляет собой неменяющееся (медленно меняющееся) изображение, на котором происходит действие игры. Поле игры изображается отдельными программами, загружаемыми в начале игры, поэтому нет необходимости загружать все программы верхнего уровня для динамической генерации игрового поля. Такой подход описан в данной главе. Программы, используемые для генерации изображений, их хранения в файле на диске и доступа к ним, описаны в главе 4. Они загружают эти изображения по мере необходимости.

МУЛЬТИПЛИКАЦИЯ НА ЭКРАНЕ

Ключевым и наиболее впечатляющим моментом видеоигры является мультипликация. Мультипликация - основной отличительный признак видеоигр. Основной метод мультипликации прост: уничтожить изображение предмета и создать его вновь, но с некоторым небольшим смещением. Скорость этого процесса должна быть очень высокой. Это может быть обеспечено путем непосредственного доступа к видеопамяти дисплея, возможность которого описана в главе 4.

Для повышения качества изображения, быстродействия операций уничтожения и повторного изображения объекта используется операция "НЕ-ИЛИ" для двоичного кода каждой точки объекта на экране. Этот способ обеспечивает возможность быстрого перемещения спрайта по экрану, не меняя его цвет и размеры, и фактически не уничтожая в памяти терминала данные о его изображении.

Программа, отображающая на экране терминала спрайт, представляет собой некоторую модификацию функции display_object() из главы 4.

```
/* отображение объекта на экране */
void display_object(ob, sides,cc)
double ob[][4];
int sides,cc;
{
  register int i;
  for(i=0; i<sides; i++)
    line((int)ob[i][0], (int)ob[i][1],
         (int)ob[i][2], (int)ob[i][3], cc | 128);
}
```

Как вы могли убедиться, функция display_object() рисует все линии объекта, используя приведенную в главе 4 функцию line(). Заметим, что значение номера цвета складывается по схеме "ИЛИ" с

числом 128 в команде установки старших битов. Это приводит к тому, что в функции mempoint(), используемой в функции line() для помещения изображения каждой точки, выполняется сложение по схеме "НЕ-ИЛИ" двоичного кода. Это позволяет спрайту всегда оставаться видимым независимо от собственного цвета и цвета фона.

Для демонстрации мультипликации введите в ваш компьютер следующую программу. Эта программа позволит вам перемещать спрайт (в виде маленького крестика размером 6x6 точек растра) по экрану, используя клавиши управления курсором. Если ваш компьютер не включает функцию bioskey(), то просмотрите главу 1 для определения версии компилятора, которая вам необходима.

```
#include "dos.h"
#include "stdio.h"
void mode(), line();
void mempoint(), palette();
void display_object(),update_object();
unsigned char read_point();
int sprite[2][4] = {
   3,0,3,5,
   0,3,5,3
};
main()
{
     union k {
        char c[2];
        int i;
     } key;
   int deltax=0,deltay=0;
   mode(4); /*m установка 4 режима графики CGA/EGA */
   palette(0); /* палитра 0 */
   display_object(sprite,2,1);
   do {
          key.i = bioskey(0);
          deltax=0;deltay=0;
          if(!key.c[0]) switch(key.c[1]) {
             case 75: /* влево */
                deltay= -1;
                 break;
             case 77: /* вправо */
                deltay= 1;
                 break;
             case 72: /* вверх */
                deltax= -1;
                break;
             case 80: /* вниз */
                deltax= 1;
                break;
             case 71: /* вверх и влево */
                deltay= -1;
                deltax= -1;
                break;
             case 73: /* вверх и вправо */
                deltay= 1;
                deltax= -1;
                break;
             case 79: /* вниз и влево */
                deltay= -1;
                deltax= 1;
                break;
             case 81: /* вниз и вправо */

                deltay= 1;
                deltax= 1;
                break;
```

```
            }
            /* стирание текущей позиции спрайта */
            display_object(sprite,2,1);
            if(is_legal(sprite,deltax,deltay,2))
            update_object(sprite,deltax,deltay,2);
            /* перезапись спрайта в новую позицию */
            displey_object(sprite2,1);
        }   while (key.c[0]!='q');
        getchar();
        mode(2);
     }
    /* Выбор палитры */
    void palette(pnum)
    int pnum;
      {
      union REGS r;
      r.h.bh = 1; /* код 4-го графического режима */
      r.h.bl = pnum;
      r.h.ah = 11;
      int86(0x10, &r, &r);
      }
    /* Выбор режима */
    void mode(mode_code)
    int mode_code;
      {
      union REGS r;
      r.h.al = mode_code;
      r.h.ah = 0;
      int86(0x10, &r, &r);
      }
/* Изображение линии заданного цвета с использованием
   алгоритма Брезенхама */
 void line(startx,starty,endx,endy,color)
 int startx,starty,endx,endy,color;
 {
   register int t,distance;
   int x=0,y=0,delta_x,delta_y;
   int incx,incy;
 /* Вычисление расстояния в обоих направлениях         */
   delta_x=endx-startx;
   delta_y=endy-starty;
 /* определение направления шага,
    шаг вычисляется либо по вертикальной, либо по горизонтальной
    линии                    */

    if(delta_x>0) incx=1;
    else  if(delta_x==0) incx=0;
    else  incx= -1;
    if(delta_y>0) incy=1;
    else  if(delta_y==0) incy=0;
    else  incy= -1;
  /* определение какое расстояние больше */
    delta_x=abs(delta_x);
    delta_y=abs(delta_y);
    if(delta_x>delta_y) distance=delta_x;
    else distance=delta_y;
  /* Изображение линии */
    for (t=0; t<=distance+1; t++) {
       mempoint(startx,starty,color);
       x+=delta_x;
       y+=delta_y;
       if(x>distance) {
          x-=distance;
          startx+=incx;
```

```
      }
      if(y>distance) {
         y-=distance;
         starty+=incy;
      }
   }
}
 /* Запись точки в CGA/EGA */
  void mempoint(x,y,color_code)
  int x,y,color_code;
  {
     union mask {
        char c[2];
        int i;
     } bit_mask;
     int i,index,bit_position;
     unsigned char t;
     char xor; /* "исключающее ИЛИ" цвета в случае его
                  изменения */
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                              памяти CGA */
     bit_mask.i=0xFF3F; /* 11111111 00111111 в
                           двоичном виде */
     if(x<0 || x>199 || y<0 || y>319) return;
     xor=color_code & 128; /* проверка, устанавливался ли
                              режим "исключающего ИЛИ" */
     color_code=color_code & 127; /* маска старших битов */
     /*  установка битовой маски и битов режима цвета
         в правую позицию */

      bit_position=y%4; /* вычисление нужной позиции
                           в байте */
      color_code<<=2*(3-bit_position); /* сдвиг кода цвета
                                    в нужную позицию */
      bit_mask.i>>=2*bit_position; /* сдвиг битовой маски в
                                      нужную позицию */
      /* определение требуемого байта в памяти терминала */
       index=x*40+(y%4);
       if(x%2) index+=8152; /* если нечетный, используется
                               второй блок */
       /* запись цвета */
        if(!xor) {  /* режим изменения цвета */
           t=*(ptr+index) & bit_mask.c[0];
           *(ptr+index)=t|color_code;
        }
        else {
           t=*(ptr+index) | (char)0;
           *(ptr+index)=t & color_code;
        }
     }
 /* чтение байта из оперативной памяти CGA/EGA */
  unsigned char read_point(x,y)
  int x,y;
  {
     union mask {
        char c[2];
        int i;
     } bit_mask;
     int i,index,bit_position;
     unsigned char t;
     char xor; /* "исключающее ИЛИ" цвета в случае его
                  изменения */
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                              памяти CGA */
```

```
  bit_mask.i=3; /* 11111111 00111111 в
                        двоичном виде */
  if(x<0 || x>199 || y<0 || y>319) return 0;
  /*  установка битовой маски и битов режима цвета
      в правую позицию */
   bit_position=y%4; /* вычисление нужной позиции
                        в байте */
   bit_mask.i<<=2*(3-bit_position);
   /* определение требуемого байта в памяти терминала */
    index=x*40+(y>>4);
    if(x%2) index+=8152; /* если нечетный, используется
                            второй блок */

    /* запись цвета */
     t=*(ptr+index) & bit_mask.c[0];
     t>>=2*(3-bit_position);
     return t;
  }
/* отображение объекта на экране */
void display_object(ob, sides,cc)
double ob[][4];
int sides,cc;
{
  register int i;
  for(i=0; i<sides; i++)
    line((int)ob[i][0], (int)ob[i][1],
         (int)ob[i][2], (int)ob[i][3], cc|128);
}
/* Смещение (параллельный перенос) объекта в направлении,
определенном x и y
*/
void update_object(ob, x, y, sides)
int ob[][4];        /* объект */
int x, y;           /* направление смещения */
register int sides; /* количество сторон объекта */
  {
  sides--;
  for(; sides>=0; sides--)
    {
    ob[sides][0] += x;
    ob[sides][1] += y;
    ob[sides][2] += x;
    ob[sides][3] += y;
    }
  }
/* Определение допустимости перемещения объекта.
Возвращает 1, если перемещение допустимо, 0- в
противном случае
*/
void is_legal(ob, x, y, sides)
int ob[][4];          /* объект */
int x, y;             /* шаг перемещения */
int sides;            /* число сторон объекта */
  {
  if(x==0 && y==0)
    return 1;      /* пустое перемещение всегда допустимо*/
  sides--;
  for(; sides>=0; sides--)
    {
    /* контроль выхода за допустимую область */
    if(ob[sides][0]+x>199 || ob[sides][1]+y>319)

      return 0;
    if(ob[sides][2]+x<0 || ob[sides][3]+y<0)
```

```
          return 0;
        }
       return 1;
       }
```

Рассмотрим кратко, как работает эта программа. Клавиши управления курсором (клавиши со стрелками и клавиши <HOME>, <PGUP>, <END> и <PGDN>) определяют положение спрайта. При нажатии клавиши спрайт смещается на одну точку растра в указанном направлении. Клавиши-стрелки управляют горзонтальными и вертикальными перемещениями, остальные - диагональными. Функция is_legal() определяет возможность дальнейшего перемещения спрайта в выбранном направлении. Если возможен выход спрайта за пределы границ экрана, то такое перемещение запрещается. Все остальные функции этой программы работают, как описано в главе 4.

Обычно необходимо сохранять размер объекта, который вы "оживляете" (особенно небольшого) для того, чтобы его можно было перерисовывать с высокой скоростью. Это обеспечивает плавность движения при мультипликации. Если объект достаточно большой, то его движение будет дискретно. При разработке видеоигр необходимо так подбирать размеры спрайта, чтобы возможности компьютера и адаптера реализовывались оптимальным образом.

МУЛЬТИПЛИКАЦИЯ СПРАЙТА

Передвижение спрайта по экрану составляет только половину возможностей его "оживления". В основном спрайт будет использоваться на экране для того, чтобы создавать иллюзию движения. Например, спрайт, который выглядит подобно человеку, может передвигать ногами, как будто он идет. Этот тип "оживления" является наиболее впечатляющим (и наиболее легким). Для обеспечения такой возможности разрабатываются два или более вариантов спрайта, отличие между которыми заключается в том, что некоторые из частей спрайта отличаются от первоначального его варианта. Программа последовательно меняет варианты спрайта в процессе его движения по экрану.

В качестве примера изменим программу main(), как это показано ниже, и добавим в нее второй спрайт. Второй спрайт отображает крестик ("+"), повернутый под углом в 45 градусов. Если вы запустите программу, то будет создаваться впечатление, что крестик вращается в процессе передвижения по экрану. Переменная swap используется для выбора типа текущего спрайта.

```
int sprite2[2][4] = {
   0,0,5,5,
   0,5,5,0
};
main()
{
     union k {
        char c[2];
        int i;
     } key;
   int deltax=0,deltay=0; /* направление движения */
   int swap=0; /* тип спрайта */
   mode(4); /* установка 4 режима графики CGA/EGA */
   palette(0); /* палитра 0 */
   display_object(sprite,2,1);
   do {
          key.i = bioskey(0);
          deltax=0;deltay=0;
          if(!key.c[0]) switch(key.c[1]) {
             case 75: /* влево */
                deltay= -1;
                 break;
             case 77: /* вправо */
```

```
                deltay= 1;
                 break;
              case 72: /* вверх */
                deltax= -1;

                break;
              case 80: /* вниз */
                deltax= 1;
                break;
              case 71: /* вверх и влево */
                deltay= -1;
                deltax= -1;
                break;
              case 73: /* вверх и вправо */
                deltay= 1;
                deltax= -1;
                break;
              case 79: /* вниз и влево */
                deltay= -1;
                deltax= 1;
                break;
              case 81: /* вниз и вправо */
                deltay= 1;
                deltax= 1;
                break;
             }
            /* стирание текущей позиции спрайта */
            if(!swap) displey_object(sprite,2,1);
            else displey_object(sprite2,2,1);
            if(is_legal(sprite,deltax,deltay,2)) {
               update_object(sprite,deltax,deltay,2);
               update_object(sprite2,deltax,deltay,2);
            }
            swap= !swap; /* смена типа спрайта */
            /* перезапись спрайта в новую позицию */
            if(!swap) displey_object(sprite,2,1);
            else displey_object(sprite2,2,1);
        }   while (key.c[0]!='q');
        getchar();
        mode(2);
     }
```

ОРГАНИЗАЦИЯ ДАННЫХ В ВИДЕОИГРАХ

Подобно остальным программам, программы видеоигр включают как операторы, так и данные. Кроме счета игры и статуса различных, расходуемых в процессе игры ресурсов (например, количество запущенных фотонных торпед), большинство данных, используемых в видеоиграх, представляют собой позиции экрана для различных объектов. Координаты позиций экрана для движущихся объектов должны храниться в установленных переменных. Информацию о фиксированных объектах игрового поля целесообразно хранить непосредственно в видеопамяти терминала. Если в процессе игры потребуется информация для изменения игрового поля (как это часто бывает), осуществляется доступ к видеопамяти и оттуда считываются массивы с информацией об измененном объекте.

Контрооль границ.

В большинстве видеоигр существуют спрайты, которые находятся под управлением пользователя. Обычно игроку не разрешается перемещать спрайт через некоторые объекты игрового поля или через другой спрайт. Есть два способа ограничения местонахождения спрайта. В первом способе в установленных переменных хранятся

граничные точки области, где разрешено движение спрайта. При передвижении спрайта по экрану осуществляется контроль на выход за пределы этих допустимых значений. Однако этот метод обладает довольно малой реактивностью и для игр с большим количеством объектов неэффективен. Более удобным способом является простая проверка области экрана на предмет нахождения в ней какого-либо объекта путем контроля соответствующей области видеопамяти. Это обеспечивается тем, что информация об игровом поле уже находится в видеопамяти и бессмысленно ее где-либо дублировать.

Изменение цвета.

В процессе игры удобно оперировать объектами разного цвета. Например, красный цвет может отображать границы непересекаемых областей, зеленый цвет используется для вашего спрайта, а желтый - для спрайта противника. Хотя это можно сделать с использованием переменных, описывающих эти объекты, часто бывает удобнее заранее определять для объекта его цвет. Это не только упростит процесс программирования видеоигры, но и сделает ее более быстродействующей. Например, если в пурпурный цвет окрашена мина, то считается, что вы на ней подорвались лишь в том случае, если одна из точек вашего спрайта окрашивается в пурпурный цвет.

Программирование в цвете видеоигр имеет длинную историю. Например, первая игра "пинг-понг" имела только два цвета: белый и черный. В этой игре белый цвет был несовместим с белым (они отталкивались), но можно было двигаться по черному игровому полю. Таким образом, белый шарик мог перемещаться по черному полю, если ударялся белой ракеткой или отражался от белой стены (линии) позади ракетки. Эти основные принципы использовались и тогда, когда в игре стали появляться и другие цвета. После того, как объекты видеоигр стали программироваться в цвете, разработка программ обработки игровых ситуаций значительно упростилась и увеличилась скорость их работы.

ТАБЛО СЧЕТА АКТИВНОГО ПРОТИВНИКА

Роль компьютера в игре во многом зависит от того, является ли эта игра для одного человека или для двоих. Если в игре участвуют два человека, компьютеру отводится роль арбитра и функции табло для отображения счета. Однако, в игре, где участвует один игрок, компьютер становится активным противником. С точки зрения программирования разработка игр, где компьютер выступает в роли противника, значительно интересней.

РАЗРАБОТКА ВИДЕОИГРЫ

В этом параграфе мы опишем разработку видеоигры, которая иллюстрирует многие принципы, описанные в данной главе.

Описание игры

Первым шагом в процессе создания видеоигры является определение ее природы и правил, по которым она ведется. Программа, описанная здесь, представляет собой компьютеризованную версию традиционной детской игры "салочки". Игрок и компьютер управляют каждый своим "человеком". Один из них догоняет другого, и, если у них произошел контакт, происходит смена амплуа. Победителем в игре становится тот, кто больший промежуток времени был в положении догоняемого.

Счет определяется путем фиксации игрового времени: после каждой прошедшей секунды добавляется одно очко тому, кто находится в роли догоняемого. Счет непрерывно отображается в углу экрана. Игра заканчивается, когда один из игроков набирает 999 очков. Для удобства игра может быть закончена путем нажатия

клавиши <Q>.

Игрок управляет движением спрайта посредством клавиш управления курсором. Игровое поле в данном случае не создается самой программой игры. Для этих целей используются программы рисования ("программы-художники"), описанные в главе 4. Поэтому, вы можете самостоятельно создавать различную среду игры.

Использование цвета и граничные условия.

Игра "салочки" использует программирование в цвете для идентификации различных объектов. Например, вы можете сделать спрайт игрока зеленым, спрайт компьютера - желтым, а границы поля игры - красными. В соответствии с этим подходом, нет необходимости хранить отдельные массивы данных в разделяемой области программы, т.к. подпрограммы могут просто контролировать содержимое видеопамяти. В данном случае, так же, значительно упрощается процесс ограничения области движения спрайтов красной линией. Для этой цели необходимо немного изменить функцию is_legal(), описанную ранее, как это показано ниже.

```
/* Определение допустимости перемещения объекта.
Возвращает 1, если перемещение допустимо, 0 - в
противном случае
*/
void is_legal(ob, x, y, sides)
int ob[][4];          /* объект */
int x, y;             /* шаг перемещения */
int sides;          /* число сторон объекта */
  {
  if(x==0 && y==0)
    return 1;    /* пустое перемещение всегда допустимо*/
  sides--;
  for(; sides>=0; sides--)
    {
    /* контроль выхода за допустимую область */
    if(ob[sides][0]+x>199 || ob[sides][1]+y>319)
      return 0;
    if(ob[sides][2]+x<0 || ob[sides][3]+y<0)
      return 0;
    /* контроль препятствий */
    if(read_point(ob[sides][0]+x, ob[sides][1]+y)==2)
      return 0;
    if(read_point(ob[sides][2]+x, ob[sides][3]+y)==2)
      return 0;
    }
  return 1;
  }
```

Напомним коды различных цветов: желтый - 1, красный - 2, зеленый - 3, черный (фон) - 0.

Описание спрайта.

Спрайт, используемый в данной игре, будет похож на бегущего человека.

Вид спрайта на стр. 187 не может быть воспроизведен имеющимися средствами. (Ред. пер. И.Бычковский.)

В программе существует два типа спрайта. Во втором спрайте ноги "человека" сжаты вместе. Быстрая смена между изображениями двух типов спрайта создает иллюзию бегущего человека.

Спрайт пользователя начинает игру в верхнем левом углу экрана, а спрайт компьютера - в нижнем правом. Описание спрайтов приведено ниже.

```
int human[4][4] = /* это ваш спрайт */
```

```
        {
        1,      6,      6,      6,
        4,      2,      3,      9,
        9,      1,      6,      6,
        9,      11,     6,      6
        };
    int human2[4][4] =
        {
        1,      6,      6,      6,
        4,      2,      3,      9,
        9,      3,      6,      6,
        9,      9,      6,      6
        };
    int computer[4][4] = /* это  спрайт компьютера */
        {
        180,      6,      185,      6,
        183,      2,      182,      9,
        188,      1,      185,      6,
        188,      11,     185,      6
        };
    int computer2[4][4] =
        {
        180,      6,      185,      6,
        183,      2,      182,      9,
        188,      3,      185,      6,
        188,      9,      185,      6
        };
```

ТЕЛО ГЛАВНОЙ ПРОГРАММЫ

После того как вы научитесь разрабатывать собственные видеоигры, вы поймете, что все они имеют одну главную общую деталь - программу, управляющую игрой. Алгоритм таких программ довольно-таки сходен для различных видеоигр. Главная программа генерирует движение объектов по экрану, контролирует нажатие клавиш пользователем и реагирует на них, проверяет допустимость заданных перемещений, подсчитывает набранные очки и отображает счет, последовательно вызывает функции отображения объектов на экране.

```
    int directx,directy; /* направление  */
    main()
    {
       union k {
          char c[2];
          int i;
       } key;
     int deltax=0,deltay=0;
     int swaph=0,swapc=0;
     int it=COMPUTER;
     long htime,ctime,starttime,curtime; /* таймер счета */
     int count;
     mode(4); /* установка 4 режима графики CGA/EGA */
     palette(0); /* палитра 0 */
     load_pic(); /* ввод игрового поля */
     time(&starttime); /* установка времени */
     htime=ctime=0;
     display_object(human,4,1);
     display_object(computer,4,3);
     count=0;
     /* главный цикл игры */
     do {
         /* вычисление текущего счета  */
        time(&curtime);
        if(it==COMPUTER) htime+=curtime-starttime;
```

```
else ctime+=curtime-starttime;
time(&starttime);
show_score(it,htime,ctime);
if(bioskey(1)) { /* если нажата клавиша */
    directx=directy=IDLE; /* устанавливает
                 направление перемещения */
    key.i = bioskey(0);
    deltax=0;deltay=0;

    if(!key.c[0]) switch(key.c[1]) {
       case 75: /* влево */
          deltay= -1;
          directy=LEFT;
          break;
       case 77: /* вправо */
          deltay=1;
          directy=RIGHT;
          break;
       case 72: /* вверх */
          deltax= -1;
          directx=UP;
           deltax= -1;
          directx=UP;
          break;
       case 80: /* вниз */
          deltax=1;
          directx=DOWN;
          break;
       case 71: /* вверх и влево */
          deltay= -1;
          directy=LEFT;
          deltax= -1;
          directx=UP;
          break;
       case 73: /* вверх и вправо */
          deltay=1;
          directy=RIGHT;
          deltax= -1;
          directx=UP;
          break;
       case 79: /* вниз и влево */
          deltay= -1;
          directy=LEFT;
          deltax=1;
          directx=DOWN;
          break;
       case 81: /* вниз и вправо */
          deltay=1;
          directy=RIGHT;
          deltax=1;
          directx=DOWN;
          break;
     }
   }
   /* смена типа спрайта игрока  */
   if(!swaph) display_object(human,4,1);
   else display_object(human2,4,1);
   if(is_legal(human,deltax,deltay,4)) {
      update_object(human,deltax,deltay,4);
      update_object(human2,deltax,deltay,4);
   }
   /* проверяет: попался ли убегающий */
   if(!count && tag(human,computer)) {
```

```
            it= !it; /* смена амплуа */
            count=6;
          }
          swaph= !swaph; /* смена фигуры имитирующей бег */
          /* изображение "человека" в новой позиции */
          if(!swaph) displey_object(human,4,1);
          else displey_object(human2,4,1);
          if(!swapc) display_object(computer,4,3);
          else display_object(computer2,4,3);
          /* генерация движения спрайта компьютера */
          if(it==COMPUTER)
          it_comp_move(computer,computer2,human,4);
          else
        not_it_comp_move(computer,computer2,directx,directy,4);
          if(!count && tag(human,computer)) {
             it= !it;
             count=6;
        /* компьютер догоняет; изменение координаты X на 2
           так, чтобы быстрей стать догоняемым
        */
              if(is_legal(computer, 2, 0, 4))
              {
                 update_object(computer, 2, 0, 4);
                update_object(computer2, 2, 0, 4);
              }
             else
             {
               update_object(computer, -2, 0, 4);
               update_object(computer2, -2, 0, 4);
             }
          }
          swapc = !swapc; /* заменить тип спрайта */
      /* вывод на экран спрайта компьютера */
          if(!swapc) display_object(computer, 4, 3);
          else       display_object(computer2, 4, 3);
          if(count) count--;
       }
       while (key.c[0] !='q' && htime<999 && ctime<999);
       mode(2);
       if(ctime>htime)
          printf("Компьютер выиграл!");
       else
          printf("Вы победили!");
    }
```

В теле главной программы экран терминала устанавливается в 4-й графический режим, выбирается палитра 0 и инициализируются переменные счета игры. После этого оба спрайта отображаются в своих исходных позициях.

Переменная htime содержит значение счета игрока, а ctime - компьютера. Переменные swapc и swaph предназначены для указания типа спрайта. Переменные deltax и deltay содержат изменения значений координат после очередного нажатия клавиш игроком. Глобальные переменные directx и directy содержат координаты спрайта, управляемого игроком. Значения этих величин используются компьютером для генерации перемещения своего спрайта. Переменная-признак it содержит информацию о том, кто в данный момент находится в режиме догоняющего. Она может принимать одно из двух значений, описанных в макроопределении **#define**: COMPUTER или HUMAN.

Главная программа работает циклически. На экране отображается текущий счет. После этого проверяется, была ли нажата какая-либо клавиша. Если клавиша была нажата, то определяется ее код и производится заданное перемещение спрайта

игрока. Обратите внимание на то, что в данной программе нет режима ожидания нажатия клавиши игроком. Поэтому, не смотря на то, что игрок не нажимает клавиш, компьютер продолжает свою работу, и спрайт игрока продолжает указанное перемещение до тех пор, пока не будет нажата другая клавиша. Такое движение спрайта обеспечивает достаточно высокую динамичность игры.

После очередного перемещения спрайта игрока, компьютер генерирует движение собственного спрайта, если это необходимо. Обратите внимание на то, что для генерации движения спрайта используются различные функции в зависимости от того в режиме догоняющего или догоняемого находится спрайт. При реализации очередного перемещения собственного спрайта, компьютер так же проверяет его корректность.

Рассмотрим некоторые программы, используемые в этой игре.

Программа генерации движения спрайта компьютера.

Если спрайт компьютера находится в режиме догоняющего, то для генерации очередного его кванта движения используется функция it_comp_move**()**. В основном компьютер повторяет стратегию движения пользователя. Движение его спрайта отклоняется из-за того, что он должен обходить объекты-препятствия. Однако спрайт компьютера может игнорировать некоторые объекты, что позволяет выровнять баланс игры.

Приведем текст функции it_comp_move**()**.

```
/* Генерация движения спрайта компьютера, когда
   он в роли догоняющего */
void it_comp_move(ob1, ob2, human, sides)
int ob1[][4], ob2[][4], human[][4], sides;
     {
     register int x, y, d; /* d = direction */
     static skip = 0;
     skip++;
     if(skip==3)
       {
       skip=0;
       return;
       /* уменьшение времени реакции компьютера */
       }
     x = 0;
     y = 0;
/* движение к игроку */
  if(human[0][0]<ob1[0][0])
       x = -1;
     else
       if(human[0][0]>ob1[0][0])
         x = 1;
     if(human[0][1]<ob1[0][1])
       y = -1;
     else
       if(human[0][1]>ob1[0][1])
         y = 1;
     if(is_legal(ob1, x, y, sides))
       {
       update_object(ob1, x, y, sides);
       update_object(ob2, x, y, sides);
       }
     else
       {
       if(x && is_legal(ob1, x, 0, sides))
         {
         update_object(ob1, x, 0, sides);
         update_object(ob2, x, 0, sides);
```

```
            }
          else
            if(is_legal(ob1, 0, y, sides))
            {
            update_object(ob1, 0, y, sides);
            update_object(ob2, 0, y, sides);
            }
          }
        }
```

Заметим, что эта функция меняет положение спрайта в 3 раза медленнее, чем это возможно. Делается такое замедление с целью снижения быстродействия компьютера до уровня человека.

Функция, генерирующая движение спрайта в режиме догоняемого, обеспечивает движение в сторону, противоположную от спрайта игрока. Хотя этот алгоритм является неоптимальным, он делает игру достаточно привлекательной и требует от пользователя хорошей реакции.

```
     /* Генерация движения спрайта компьютера, когда
        он выступает в роли убегающего */
     void it_comp_move(ob1, ob2, human, sides)
     int ob1[][4], ob2[][4], human[][4], sides;
          {
          register int x, y, d; /* d = direction */
          static skip = 0;
          skip++;
          if(skip==3)
            {
            skip=0;
            return;
            /* уменьшение времени реакции компьютера */
            }
          x = 0;
          y = 0;
   /* движение к игроку */
     if(human[0][0]<ob1[0][0])
            x = -1;
          else
            if(human[0][0]>ob1[0][0])
              x = 1;
          if(human[0][1]<ob1[0][1])
            y = -1;
          else
            if(human[0][1]>ob1[0][1])
              y = 1;
          if(is_legal(ob1, x, y, sides))
            {
            update_object(ob1, x, y, sides);
            update_object(ob2, x, y, sides);

            }
          else
            {
            if(x && is_legal(ob1, x, 0, sides))
              {
              update_object(ob1, x, 0, sides);
              update_object(ob2, x, 0, sides);
              }
            else
              if(is_legal(ob1, 0, y, sides))
              {
              update_object(ob1, 0, y, sides);
              update_object(ob2, 0, y, sides);
              }
            }
```

```
     }
/* генерация движения спрайта компьютера, когда
   он убегает     */
void not_it_comp_move(ob1, ob2, dx, dy, sides)
int ob1[][4], ob2[][4];
int dx, dy; /* направление последнего перемещения
               "человека" */
int sides;
     {
     register int x, y, d;
     static skip = 1;
     skip++;
     if(skip==3)
       {
       skip = 0;
       return;
       /* уменьшение времени реакции компьютера в 3 раза */
       }
     x = 0;
     y = 0;
     /* перемещение в противоположном направлении */
     x = -dx;
     y = -dy;
     if(is_legal(ob1, x, y, sides))
       {
       update_object(ob1, x, y, sides);
       update_object(ob2, x, y, sides);
       }
     else
       {
       if(x && is_legal(ob1, x, 0, sides))
         {
         update_object(ob1, x, 0, sides);
         update_object(ob2, x, 0, sides);
         }
       else if(is_legal(ob1, 0, y, sides)) {

           update_object(ob1, 0, y, sides);
           update_object(ob2, 0, y, sides);
           }
         }
       }
```

Эта функция так же как и предыдущая, работает с 3-кратным замедлением.

Программа контроля касания спрайтов.

В этой игре режимы спрайтов изменяются на противоположные в том случае, если координаты хотя бы одной точки догоняющего спрайта совпадут с координатами любой точкой догоняемого. Правила игры могут быть изменены таким образом, что изменение режима произойдет лишь в случае полного совмещения спрайтов. Но эта довольно-таки сложная задача для многих игроков. Приведенная ниже функция tag() возвращает значение 1, если спрайты столкнулись, и 0 - в противном случае.

```
/* Проверяет есть ли контакт между спрайтами */
tag(ob1, ob2)
int ob1[][4], ob2[][4];
     {
     register int i;
     /* для смены амплуа необходимо, чтобы спрайты
        имели хотя бы одну общую точку растра */
     for (i= -1; i<2; i++)
       if(ob1[0][0]==ob2[0][0]+i && ob1[0][1]==ob2[0][2]+i)
```

```
            return 1;
          return 0;
          }
```

Вы можете внести изменения в функцию tag() и установить свои правила контроля режимов спрайтов.

Полный текст программы игры TAG.

В данном разделе приведен текст программы игры TAG, похожей на русские "салочки". Вы можете ввести ее в свой компьютер, если он снабжен графическим адаптером.

```
  /* Пример мультипликации игры "салочки"
     Объектом в игре является "человек", который
     догоняет другого "человека".
     Ваш "человек"- зеленый,"человек" компьютера-
     желтый. Все, что окрашено в красный цвет,
     пересекать нельзя.
     Для смены ролей догоняющего и догоняемого
     необходимо, чтобы "люди" пересеклись
     хотя бы в одной точке растра  */
   #define COMPUTER 0
   #define HUMAN 1
   #define IDLE 0
   #define DOWN 1
   #define UP -1
   #define LEFT -1
   #define RIGHT 1
   #include "dos.h"
   #include "stdio.h"
   #include "math.h"
   #include "time.h"
   void mode(), line();
   void mempoint(), palette(), xhairs();
   void goto_xy(),show_score();
   void display_object(),update_object();
   void it_comp_move(),not_it_comp_move();
   void save_pic(), load_pic();
   unsigned char read_point();
   int human[4][4] = { /* ваш спрайт */
      1,6,6,6,
      4,2,3,9,
      9,1,6,6,
      9,11,6,6
    };
    int human2[4][4] = {
      1,6,6,6,
      4,2,3,9,

      9,3,6,6,
      9,9,6,6
     };
     int computer[4][4] = { /* спрайт компьютера */
       180,6,185,6,
       183,2,182,9,
       188,1,185,6,
       188,11,185,6
     };
     int computer2[4][4] = {
       180,6,185,6,
       183,2,182,9,
       188,3,185,6,
       188,9,185,6
     };
     int directx,directy; /* направление  */
```

```
main()
{
   union k {
      char c[2];
      int i;
   } key;
 int deltax=0,deltay=0;
 int swaph=0,swapc=0;
 int it=COMPUTER;
 long htime,ctime,starttime,curtime;
 int count;
 mode(4); /* установка 4 режима графики CGA/EGA */
 palette(0); /* палитра 0 */
 load_pic(); /* ввод игрового поля */
 time(&starttime); /* установка времени */
 htime=ctime=0;

 display_object(human,4,1);
 display_object(computer,4,3);
 count=0;
 /* главный цикл игры */
 do {
     /* вычисление текущего счета  */
    time(&curtime);
    if(it==COMPUTER) htime+=curtime-starttime;
    else ctime+=curtime-starttime;
    time(&starttime);
    show_score(it,htime,ctime);
    if(bioskey(1)) { /* если нажата клавиша */

        directx=directy=IDLE; /* устанавливает
                    направление перемещения */
        key.i = bioskey(0);
        deltax=0;deltay=0;
        if(!key.c[0]) switch(key.c[1]) {
           case 75: /* влево */
              deltay= -1;
              directy=LEFT;
              break;
           case 77: /* вправо */
              deltay=1;
              directy=RIGHT;
              break;
           case 72: /* вверх */
              deltax= -1;
              directx=UP;
               deltax= -1;
              directx=UP;
              break;
           case 80: /* вниз */
              deltax=1;
              directx=DOWN;
              break;
           case 71: /* вверх и влево */
              deltay= -1;
              directy=LEFT;
              deltax= -1;
              directx=UP;
              break;
           case 73: /* вверх и вправо */
              deltay=1;
              directy=RIGHT;
              deltax=-1;
              directx=UP;
```

```
            break;
          case 79: /* вниз и влево */
             deltay= -1;
             directy=LEFT;
             deltax=1;
             directx=DOWN;
             break;
          case 81: /* вниз и вправо */
             deltay=1;
             directy=RIGHT;
             deltax=1;
             directx=DOWN;
             break;
        }
      }
      /* смена типа спрайта игрока  */
      if(!swaph) display_object(human,4,1);
      else displey_object(human2,4,1);
      if(is_legal(human,deltax,deltay,4)) {
         update_object(human,deltax,deltay,4);

         update_object(human2,deltax,deltay,4);
      }
      /* проверяет: попался ли убегающий */
      if(!count && tag(human,computer)) {
         it=!it; /* смена амплуа */
         count=6;
      }
      swaph= !swaph; /* смена фигур, имитирующих бег */
      /* вывод "человека" в новой позиции */
      if(!swaph) display_object(human,4,1);
      else display_object(human2,4,1);
      if(!swapc) display_object(computer,4,3);
      else display_object(computer2,4,3);
      /* генерация движения спрайта компютера */
      if(it==COMPUTER)
      it_comp_move(computer,computer2,human,4);
      else
    not_it_comp_move(computer,computer2,directx,directy,4);
      if(!count && tag(human,computer)) {
         it= !it;
         count=6;
    /* компьютер догоняет; изменение координаты X на 2
       так, чтобы быстрей стать догоняемым */
          if(is_legal(computer, 2, 0, 4))
          {
             update_object(computer, 2, 0, 4);
            update_object(computer2, 2, 0, 4);
          }
        else
        {
          update_object(computer, -2, 0, 4);
          update_object(computer2, -2, 0, 4);
        }
     }
     swapc = !swapc; /* заменить тип спрайта */
/* вывод на экран спрайта компьютера */
     if(!swapc) display_object(computer, 4, 3);
     else       display_object(computer2, 4, 3);
     if(count) count--;
 }
 while (key.c[0] !='q' && htime<999 && ctime<999);
 getchar();
 mode(2);
```

```
      if(ctime>htime)
         printf("Компьютер выиграл!");
      else
         printf("Вы победили!");
    }
   /* Вывод на экран терминала счета */
   void shou_score(it, htime, ctime)
   int it;

   long htime, ctime;
     {
     goto_xy(24, 6);
     if(it==COMPUTER)
       printf("ВЫ:%ld", htime);
     else
       printf("вы:%ld", htime);
     goto_xy(24, 26);
     if(it==HUMAN)
       printf("Я:%ld", ctime);
     else
       printf("я:%ld", ctime);
     }
   /* Выбор палитры */
   void palette(pnum)
   int pnum;
     {
     union REGS r;
     r.h.bh = 1; /* код 4-го графического режима */
     r.h.bl = pnum;
     r.h.ah = 11;
     int86(0x10, &r, &r);
     }
   /* Выбор режима */
   void mode(mode_code)
   int mode_code;
     {
     union REGS r;
     r.h.al = mode_code;
     r.h.ah = 0;
     int86(0x10, &r, &r);
     }
/* изображение линии заданного цвета с использованием
   алгоритма Брезенхама */
 void line(startx,starty,endx,endy,color)
 int startx,starty,endx,endy,color;
 {
   register int t,distance;
   int x=0,y=0,delta_x,delta_y;
   int incx,incy;
 /* вычисление расстояния в обоих направлениях         */
   delta_x=endx-startx;
   delta_y=endy-starty;

 /* определение направления шага, шаг вычисляется либо   по
    вертикальной,  либо  горизонтальной  линии */
    if(delta_x>0) incx=1;
    else if(delta_x==0) incx=0; else incx=-1;

    if(delta_y>0) incy=1;
    else  if(delta_y==0) incy=0;
    else  incy=-1;
  /* определение какое расстояние больше */
    delta_x=abs(delta_x);
    delta_y=abs(delta_y);
```

```
   if(delta_x>delta_y) distance=delta_x;
   else distance=delta_y;
 /* изображение линии */
   for (t=0; t<=distance+1; t++) {
      mempoint(startx,starty,color);
      x+=delta_x;
      y+=delta_y;
      if(x>distance) {
         x-=distance;
         startx+=incx;
      }
      if(y>distance) {
         y-=distance;
         starty+=incy;
      }
   }
}
 /* запись точки в CGA/EGA */
  void mempoint(x,y,color_code)
  int x,y,color_code;
  {
     union mask {
        char c[2];
        int i;
     } bit_mask;
     int i,index,bit_position;
     unsigned char t;
     char xor; /* "исключающее ИЛИ" цвета в случае его
                  изменения */
     char far *ptr=(char far *) 0xB8000000; /* точка в
                                              памяти CGA */
     bit_mask.i=0xFF3F; /* 11111111 00111111 в
                           двоичном виде */
     if(x<0 || x>199 || y<0 || y>319) return;
     xor=color_code & 128; /* проверка, устанавливался ли
                              режим "исключающего ИЛИ" */
     color_code=color_code & 127; /* маска старших битов */
     /*  установка битовой маски и битов режима цвета
         в правую позицию */
      bit_position=y%4; /* вычисление нужной позиции

                           в байте */
      color_code<<=2*(3-bit_position); /* сдвиг кода цвета
                                          в нужную позицию */
      bit_mask.i>>=2*bit_position; /* сдвиг битовой маски в
                                      нужную позицию */
      /* определение требуемого байта в памяти терминала */
       index=x*40+(y%4);
       if(x%2) index+=8152; /* если нечетный, используется
                               второй блок */
       /* запись цвета */
        if(!xor) {  /* режим изменения цвета */
           t=*(ptr+index) & bit_mask.c[0];
           *(ptr+index)=t|color_code;
        }
        else {
           t=*(ptr+index) | (char)0;
           *(ptr+index)=t & color_code;
        }
     }
 /* чтение байта из оперативной памяти CGA/EGA */
  unsigned char read_point(x,y)
  int x,y;
  {
```

```
      union mask {
         char c[2];
         int i;
      } bit_mask;
      int i,index,bit_position;
      unsigned char t;
      char xor; /* "исключающее ИЛИ" цвета в случае его
                   изменения */
      char far *ptr=(char far *) 0xB8000000; /* точка в
                                                памяти CGA */
      bit_mask.i=3; /* 11111111 00111111 в
                           двоичном виде */
      if(x<0 || x>199 || y<0 || y>319) return 0;
      /*  установка битовой маски и битов режима цвета
          в правую позицию */
       bit_position=y%4; /* вычисление нужной позиции
                            в байте */
       bit_mask.i<<=2*(3-bit_position);
       /* определение требуемого байта в памяти терминала */
        index=x*40+(y>>4);
        if(x%2) index+=8152; /* если нечетный, используется
                                второй блок */
        /* запись цвета */

         t=*(ptr+index) & bit_mask.c[0];
         t>>=2*(3-bit_position);
         return t;
      }
/* загрузка изображения */
     void load_pic()
     {
        char fname[80];
        FILE *fp;
        register int i,j;
        char far *ptr=(char far *) 0xB8000000; /* точка в
                                                  памяти   CGA  */
        char far *temp;
        unsigned char buf[14][80]; /* содержит образ экрана */
        temp=ptr;
  /* сохранение верхних строк текущего содержимого экрана */
        for (i=0;i<14;++i)
           for (j=0;j<80;j+=2) {
              buf[i][j]=*temp;
              buf[i][j+1]=*(temp+8152);
              *temp=0; *(temp+8152)=0;/*чистка позиций экрана*/
              temp++;
           }
        goto_xy(0,0);
        printf("Имя файла:");
        gets(fname);
        if(!(fp=fopen(fname,"rb"))) {
           goto_xy(0,0);
           printf("Файл не может быть открыт\n");
           temp=ptr;
  /* восстановление содержимого экрана */
           for (i=0;i<14;++i)
              for (j=0;j<80;j+=2) {
                 *temp= buf[i][j];
                 *(temp+8125)=buf[i][j+1];
                 temp++;
              }
           return;
         }
 /* загрузка изображения из файла */
```

```
      for (i=0;i<8152;i++) {
        *ptr=getc(fp); /* четный байт */
        *(ptr+8125)=getc(fp); /* нечетный байт */
        ptr++;
      }
      fclose(fp);
    }
/* поместить курсор в заданное положение */


    void goto_xy(x,y)
    int x,y;
    {
       r.h.ah=2; /* адресация курсора */
       r.h.dl=y; /* координата столбца */
       r.h.dh=x; /* координата строки */
       r.h.bh=0; /* видео-страница */
       int86(0x10,&r,&r);
     }
   /* отображение объекта на экране */
   void display_object(ob, sides,cc)
   double ob[][4];
   int sides,cc;
   {
     register int i;
     for(i=0; i<sides; i++)
       line((int)ob[i][0], (int)ob[i][1],
            (int)ob[i][2], (int)ob[i][3], cc|128);
  }
   /* Смещение (параллельный перенос) объекта в направлении,
   определенном x и y
   */
   void update_object(ob, x, y, sides)
   int ob[][4];        /* объект */
   int x, y;           /* направление смещения */
   register int sides; /* количество сторон объекта */
     {
     sides--;
     for(; sides>=0; sides--)
       {
       ob[sides][0] += x;
       ob[sides][1] += y;
       ob[sides][2] += x;
       ob[sides][3] += y;
       }
     }
   /* Определение  допустимости перемещения объекта. Возвращает
   1, если перемещение допустимо, 0 - в противном случае */
   is_legal(ob, x, y, sides)
   int ob[][4];        /* объект */
   int x, y;           /* шаг перемещения */
   int sides;          /* число сторон объекта */
     {
     if(x==0 && y==0)
       return 1;       /* пустое перемещение всегда допустимо*/
     sides--;
     for(; sides>=0; sides--)
       {
       /* контроль выхода за допустимую область */


       if(ob[sides][0]+x>199 || ob[sides][1]+y>319)
         return 0;
       if(ob[sides][2]+x<0 || ob[sides][3]+y<0)
```

```
        return 0;
      if(read_point(ob[sides][0]+x, ob[sides][1]+y)==2)
        return 0;
      if(read_point(ob[sides][2]+x, ob[sides][3]+y)==2)
        return 0;
      }
    return 1;
    }
  /* генерация движения спрайта компьютера,  когда он догоняет
  */
    void it_comp_move(ob1,  ob2,  human, sides)
    int ob1[][4],ob2[][4], human[][4], sides;
       {
       register int x, y, d; /* d = direction */
       static skip = 0;
       skip++;
       if(skip==3)
         {
         skip=0;
         return;
         /* уменьшение времени реакции компютера */
         }
       x = 0;
       y = 0;
/* движение к игроку */
  if(human[0][0]<ob1[0][0])
         x = -1;
       else
         if(human[0][0]>ob1[0][0])
           x = 1;
       if(human[0][1]<ob1[0][1])
         y = -1;
       else
         if(human[0][1]>ob1[0][1])
           y = 1;
       if(is_legal(ob1, x, y, sides))
         {
         update_object(ob1, x, y, sides);
         update_object(ob2, x, y, sides);
         }
       else
         {
         if(x && is_legal(ob1, x, 0, sides))
           {
           update_object(ob1, x, 0, sides);
           update_object(ob2, x, 0, sides);
           }
         else


           if(is_legal(ob1, 0, y, sides))
           {
           update_object(ob1, 0, y, sides);
           update_object(ob2, 0, y, sides);
           }
         }
       }
  /* генерация движения спрайта компьютера, когда
     он убегает     */
  void not_it_comp_move(ob1, ob2, dx, dy, sides)
  int ob1[][4], ob2[][4];
  int dx, dy; /* направление последнего перемещения
               "человека" */
  int sides;
```

```
        {
        register int x, y, d;
        static skip = 1;
        skip++;
        if(skip==3)
          {
          skip = 0;
          return;
          /* уменьшение времени реакции компьютера в 3 раза */
          }
        x = 0;
        y = 0;
        /* перемещение в противоположном направлении */
        x = -dx;
        y = -dy;
        if(is_legal(ob1, x, y, sides))
          {
          update_object(ob1, x, y, sides);
          update_object(ob2, x, y, sides);
          }
        else
          {
          if(x && is_legal(ob1, x, 0, sides))
            {
            update_object(ob1, x, 0, sides);
            update_object(ob2, x, 0, sides);
            }
          else if(is_legal(ob1, 0, y, sides)) {
              update_object(ob1, 0, y, sides);
              update_object(ob2, 0, y, sides);
              }
            }
          }
    /*  проверяет наличие контакта между спрайтами */
    tag(ob1, ob2)
    int ob1[][4], ob2[][4];


        {
        register int i;
        /* для смены амплуа необходимо, чтобы спрайты
           имели хотя бы одну общую точку растра */
        for (i=-1; i<2; i++)
          if(ob1[0][0]==ob2[0][0]+i && ob1[0][1]==ob2[0][2]+i)
            return 1;
        return 0;
        }
```

Для использования игры вы должны создать одно или несколько игровых полей, используя функции, описанные в главе 4. Используйте красный цвет для изображения препятствий. Желтый и зеленый цвета можно использовать для фона. Эти цвета не несут нагрузки, поэтому могут использоваться в декоративных целях. На рисунках 5-1 и 5-2 показаны два варианта игровых полей в таком виде, в котором они отображаются на экране вашего терминала.

Быстродействие компьютеров, таких моделей как AT или PS/2 моделей 50, 60 или 80, вполне достаточно для данной игры. Темп игры будет несколько снижен на обычном компьютере PC. Однако вам уже известно, как может быть повышена динамичность игры.

Рис. 5-1 на стр. 205 имеющимися средствами воспроизведен быть не может. (Ред. пер. И.Бычковский.)

Рис. 5-1. Первое игровое поле видеоигры "салочки"

Рис. 5-2 на стр. 205 имеющимися средствами воспроизведен быть не может. (Ред. пер. И.Бычковский.)

Рис. 5-2. Второе игровое поле видеоигры "салочки"

НЕКОТОРЫЕ СООБРАЖЕНИЯ ПО ВОЗМОЖНОЙ МОДИФИКАЦИИ ПРОГРАММЫ

Возможно, вы со временем захотите создать свою видеоигру, взяв, однако, за основу рассмотренную здесь игру TAG. В этом случае в можете, например, изменить траекторию движения спрайта компьютера заставив его двигаться вокруг какого-то объекта ("охранять" его). Интересным дополнением к программе будет возможность изменять внешний вид каждого объекта-участника игры в зависимости от каких-либо условий. Кстати, решение этой задачи не требует от вас каких-либо дополнительных усилий, так как каждое изображение спрайта можно хранить в видеопамяти, а все необходимые подпрограммы для работы с ней у вас уже есть.

Другим, также представляющим интерес дополнением, может стать наделение компьютера возможностью "прогнозировать" направление движения спрайта человека. В самом деле, вы ведь знаете, куда можно двигаться, а куда нельзя, в зависимости от ситуации на экране дисплея. Так научите это делать и компьютер! Поскольку игровое поле статично, то решение и этой задачи не будет представлять сложности.

Попробуйте добавить в программу еще один спрайт, касание которого будет приносить дополнительные очки играющим.

И, наконец, последняя мысль: процесс разработки любой видеоигры начинайте с создания ее простейшего "скелета". И лишь после того, как ваш "скелет" "задышал", начинайте наращивание возможностей своей игры. Всегда стремитесь идти путем от простого к сложному.

ГЛАВА 6

ИСПОЛЬЗОВАНИЕ ПОСЛЕДОВАТЕЛЬНОГО ПОРТА: ПЕРЕДАЧА ФАЙЛОВ И ПРОСТЕЙШИЕ ЛВС

Пожалуй нет такой другой общей беды для всех программистов, как асинхронный последовательный порт. Непохожий на более простой параллельный порт, последовательный порт, как ни кто более подвержен целому семейству различных типов ошибок передачи данных. Проблема усложняется тем, что сигнал "подтверждение связи", который помогает корректно выполнять соответствующую передачу данных применительно к последовательному порту часто передается "мимо" шины кабеля, связывающего последовательный порт и внешнее устройство. Однако, несмотря на эти проблемы последовательный порт используется шире, так как именно он позволяет использовать самый дешевый путь для соединения двух устройств, разнесенных на расстояние, превышающее пару футов.

Цель этой главы - дать основы устройства последовательного порта и работы с ним, включая инициализацию, передачу и прием данных, а также обсудить наиболее общие ошибки, возникающие во время работы с последовательным портом.

Набор операций работы с последовательным портом обуславливает его использование в качестве составной части по крайней мере в двух приложениях. Во-первых, это программа пересылки файла, которая может использоваться для передачи различных типов файлов (включая двоичные файлы) между двумя компьютерами. Программа пересылки файла особенно полезна при решении проблемы стыковки различных типов компьютеров. Во-вторых, это проблема создания простейших локальных вычислительных сетей (ЛВС), включающих в себя файловый процессор (для поддержки внешних ЗУ большой емкости) и набор из двух новых команд,

позволяющих удаленным компьютерам загружать файлы из или записывать в файловый процессор.

Примеры, приведенные в этой главе, совместимы с компьютерами IBM PC, XT, AT или PS/2 (а также на совместимых с этими моделями) под управлением DOS. Однако вы легко сможете осуществить их перенос в другие операционные системы, включая OS/2.

АСИНХРОННАЯ ПОСЛЕДОВАТЕЛЬНАЯ ПЕРЕДАЧА ДАННЫХ

Перед тем, как перейти к изучению последовательного асинхронного порта вообще вам необходимо получить некоторые сведения о принципах асинхронной передачи данных. (В дальнейшем, для простоты изложения материала будем называть асинхронный последовательный порт - "последовательным портом"). Данные передаются через последовательный порт порциями в один бит за единицу времени. В этом состоит отличие последовательного порта от параллельного, который осуществляет передачу данных порциями в один байт за единицу времени. Передача данных называется асинхронной потому, что длина интервала времени между передачей очередного байта информации (по 1 биту за единицу времени) не имеет никакого значения. Поэтому основными являются синхронизация и последовательность передачи цепочки бит,которые в конечном итоге составляют байт или другую информационную единицу.

Каждый байт данных, передаваемых через последовательный порт, состоит из следующей последовательности сигнальных битов:

1. Один стартовый бит
2. Восемь битов данных (в некоторых случаях - 7)
3. Необязательный бит четности
4. Один или два конечных бита

Между передачей каждого байта может проходить некоторый промежуток времени.

Время простоя канала передачи для этого режима довольно велико. Младший бит передаваемой "порции" данных имеет нулевое значение, старший бит, завершающий очередную "порцию" данных, принимает значение равное единице. Старший бит сигнализирует о начале передачи нового байта, который считывается в канал за один цикл, начиная с младшего бита. Биты данных передаются вслед за необязательным битом четности. В конце пересылаются один или два бита, сигнализирующих о конце очередной "порции" данных, считанных за один цикл. Завершающие (конечные) биты определяют минимальное время между передачей двух байтов. Обычно число завершающих битов не имеет большого значения, поэтому вы можете использовать либо один, либо два завершающих бита в зависимости от того, какое их число используют передающий и принимающий порты.

Бит четности, если он присутствует в передаваемом сообщении, используется для контроля корректности передачи и поиска ошибок. Контроль передачи может проводиться как на четность (контрольный разряд равен сумме по модулю 2 информационных разрядов и общее число единичных разрядов четно), так и на нечетность (контрольный разряд не равен сумме по модулю 2 информационных разрядов и общее число единичных разрядов нечетно).

Скорость передачи битов по каналу измеряется в бодах (бит в секунду). Наименьшей скоростью передачи информации считается 300 бод. Эта скорость передачи использовалась в старых модемах (сейчас большинство модемов позволяют достигать скорости передачи от 1200 до 2400 бод). Семейство компьютеров IBM PC поддерживают скорость передачи данных в 9600 бод. Некоторые типы компьютеров позволяют достигать скорости передачи данных в 38400 бод!

СТАНДАРТ RS-232

Несмотря на то, что изучение стандарта RS-232 не имеет

большого влияния на понимание работы асинхронного последовательного порта в целом, ознакомление читателя со стандартом асинхронного последовательного интерфейса RS-232 (аналог в СССР - стык С-2) является целью настоящей главы. Изучение этого материала поможет вам более детально понять, какие проблемы возникают при использовании последовательного порта и как эти проблемы могут быть разрешены.

Конфигурация большинства последовательных портов является стандартной, однако наиболее широкое распространение получила конфигурация, соответствующая стандарту RS-232. По этому стандарту разъем содержит 25 контактов. (В компьютере IBM PC AT используется 9-ти контактный разъем). Следует отметить, что довольно большое число последовательных портов не поддерживают весь набор сигналов, специфицированных в стандарте RS-232. Некоторые сигналы не поддерживаются в связи с тем, что они не предназначены для использования в таком приложении и служат для других целей; другие не поддерживаются по причине того, что они выпускались в то время, когда стандарт RS-232 еще не существовал вообще или же целью их создания не являлась полная поддержка стандарта RS-232 и они в этом случае включают лишь ограниченный набор сигналов RS-232 . Наиболее общими сигналами стандарта RS-232 являются:

Сигнал	Аббревиатура	Штырь разъема
Запрос на посылку данных	RTS	4
Очистка для посылки	CTS	5
Набор данных готов	DSR	6
Набор данных завершен	DTR	20
Передача данных	TxD	2
Прием данных	RxD	3
Земля	GRD	7

На самом деле сигналов намного больше и это обусловлено тем, что последовательный порт первоначально разрабатывался как устройство поддержки модема. В связи с этим, если порт используется совместно с другими устройствами, то многие из его сигналов просто в этом случае не нужны. Эти сигналы используются для установления протокола аппаратного уровня между модемом и компьютером, если этот компьютер (1) еще не передавал информацию, но уже готов к ее передаче или (2) передача данных от модема к компьютеру еще не осуществлялась.

Ошибка кадрирования (т.е. ошибка, возникающая при передаче порции данных, передаваемой канальным уровнем сетевого взаимодействия) фиксируется в случае, если частоты синхронизирующих импульсов двух портов значительно отличаются друг от друга. Как вы можете догадаться, последовательный порт после того, как он обнаружил стартовый бит, выделяет регистр ввода, который за каждый цикл считывает один бит. Длина этого цикла определяется скоростью передачи данных. Однако время нахождения бита в регистре определяется тактовой частотой системы. Если частота компьютера-приемника недостаточна для покрытия частоты компьютера-источника, то происходит потеря полученного бита (т.к. регистр занят), в связи с чем и регистрируется ошибка кадрирования (framing error).

АППАРАТНОЕ ПОДТВЕРЖДЕНИЕ СВЯЗИ

Непосредственная передача данных из последовательного порта выполняется после того, как монитор обнаружит сигнал "очистка-для-посылки" (CTS), отправленный из порта-приемника. Вы не должны передавать данные до тех пор, пока с помощью сигнала "очистка-для-посылки" не будет индицирована надежность и безопасность передачи. Таким образом, при использовании аппаратного подтверждения связи подпрограмма передачи данных,

написанная в терминах псевдо-СИ, будет иметь вид:

```
do {
  while(not CTS) wait;
  send(byte);
} while(bytes to send);
```

Если вы имеете соединенные линией связи аппаратные средства и их сопряжение с линией связи выполнено по стандарту RS-232, то вы с успехом можете использовать те преимущества, которые вам дает аппаратное подтверждение связи. Однако совсем недавно этого нельзя было делать.

ПРОБЛЕМЫ ПЕРЕДАЧИ ДАННЫХ

При организации передачи данных с помощью модема некоторые сигналы используются для определения готовности данных или определения следующего байта посылки. Однако, когда передача данных осуществляется между двумя компьютерами, то набор сигналов (не необходимый, но желательный), используемый для обмена данными, может быть ограничен лишь сигналами GRD, TxD и RxD. Основными доводами за использование этих трех аппаратно-реализованных микропрограмм, является значительное уменьшение стоимости передачи данных по сравнению с использованием пяти или, скажем, шести микропрограмм управления. Если два компьютера одного типа соединены каналом передачи данных и один из них готов передать данные, то второй теоретически всегда готов принять их. Однако в стандарте RS-232 имеется прямо-таки настоящий ящик Пандоры, содержащий ошибки, связанные с возможностью потери или обхода сигналов протокола RS-232. Наиболее неприятными ошибками являются ошибки, связанные с переполнением регистра (overrun error).

ПЕРЕПОЛНЕНИЕ РЕГИСТРА-ПРИЕМНИКА

Если для соединения двух последовательных портов используются только три микропрограммы (сигнала), то возникает необходимость использовать своеобразный "трюк" с портом-источником в предположении, что порт-приемник уже готов к приему данных. Этот "трюк" обычно выполняется путем соединения вместе 6, 8 и 20 штырей 25-штыревого разъема. В случае неудачи эта процедура позволяет обнаружить ошибку переполнения регистра данных с большой вероятностью. Допустим теперь, что компьютер А более производительный, чем компьютер В. Если аппаратное подтверждение связи не используется, а компьютер А предполагает пересылку второго байта сообщения в компьютер В, в то время, как компьютер В выполняет чтение информации из регистра ввода данных, то будет зарегистрирована ошибка "переполнение регистра" (oberrun error). Ошибка этого типа будет также зарегистрирована даже, если компьютер В более производительный чем компьютер А, но програмное обеспечение компьютера В менее реактивно.

Эта проблема возникает потому, что штыри 6, 8 и 20 соединены и порт-источник считает, что порт-приемник всегда готов к приему данных. Короче, вы сами видите, что этот путь решения проблем является довольно сложным.

ДОСТУП К ПОСЛЕДОВАТЕЛЬНОМУ ПОРТУ КОМПЬЮТЕРА ЧЕРЕЗ BIOS

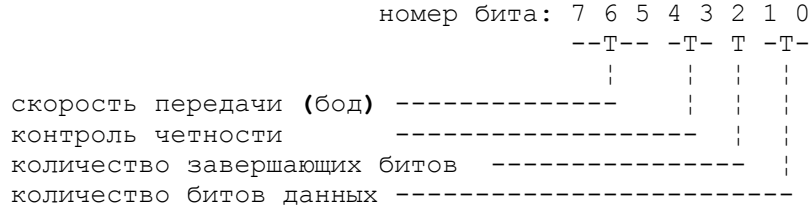
К последовательному порту компьютеров семейства PC, а также совместимых с ними моделей можно получить доступ непосредственно из DOS через ПЗУ-BIOS или в обход DOS и BIOS, используя непосредственное управление аппаратными средствами. Доступ к последовательному порту через DOS не очень хорошая идея потому, что DOS не позволяет организовать обратной связи с

последовательным портом для анализа его текущего состояния и организует лишь слепое чтение и запись данных в порт. К тому же нет возможности использовать систему прерываний DOS. Несмотря на то, что в предыдущей главе была рассмотрена возможность прямого аппаратного управления системными ресурсами, этот метод не является приемлемым для работы с последовательным портом в связи с тем, что наибольшая производительность обработки порта при использовании этого метода может быть достигнута лишь за счет прерываний ПЗУ-BIOS.

Доступ и обработку последовательного порта поддерживают четыре специальные утилиты ПЗУ-BIOS. Обработка последовательного порта осуществляется ими с помощью прерывания 14H. Разберем подробнее этот метод.

ИНИЦИАЛИЗАЦИЯ ПОРТА

Перед использованием последовательного порта вы возможно захотите установить его начальное состояние, отличающееся от принятого по умолчанию, или, другими словами, инициализировать порт. (По умолчанию, первый последовательный порт имеет следующие характеристики: скорость обмена - 1200 бод, проверка на четность, семь бит данных и один завершающий бит). Прерывание 14H, утилита 0, используется для инициализации последовательного порта. Совместно с другими прерываниями BIOS регистр AH используется для хранения номера утилиты. Регистр AL используется для хранения параметров инициализации, которые кодируются в одном байте в следующем порядке:



Скорость передачи данных кодируется в соответствии с таблицей 6-1. Контроль четности кодируется в соответствии с таблицей 6-2.

Таблица 6-1

Кодирование скорости передачи в битах 7, 6 и 5 байта инициализации последовательного порта.

Скорость	Последовательность бит
9600	1 1 1
4800	1 1 0
2400	1 0 1
1200	1 0 0
600	0 1 1
300	0 1 0
150	0 0 1
110	0 0 0

Число завершающих битов определяется значением второго разряда байта инициализации последовательного порта. Если значение этого бита равно 1, то используются два завершающих бита; в противном случае используется один завершающий бит. В конечном итоге число битов данных задается значением бит в первом и нулевом разрядах байта инициализации. Из четырех значений, которые могут устанавливаться пользователем в байте инициализации для указания числа битов данных, допустимыми являются лишь два.

Если биты в первом и нулевом разрядах байта инициализации образуют последовательность "1 0", то для передачи данных используется семь бит. Если биты в этих разрядах образуют

последовательность "1 1", то используется восемь бит данных.

Таблица 6-2

Кодирование четности в битах 4 и 3
байта инициализации последовательного порта

Вид контроля	Последовательность бит
контроль отменен	0 0 или 1 0
проверка на нечетность	0 1
проверка на четность	1 1

Например, если вы хотите установить скорость передачи данных для порта 9600 бод, проверку на четность, один завершающий бит и восемь бит для данных, вы должны установить вид байта инициализации аналогично приведенному ниже. В десятичном представлении значение байта инициализации равно 251.

```
                            1  1  1  1 1  0  1 1
                            ---T---  -T-  T  -T-
скорость передачи (бод) ------      ¦   ¦   ¦
вид контроля четности   -------------   ¦   ¦
количество завершающих битов ------------   ¦
количество битов данных ---------------------
```

Стандарт PC предусматривает наличие до семи последовательных портов (в новых типах машин их значительно больше). Для спецификации номера порта используется регистр DX. Первый последовательный порт имеет номер 0, второй - 1 и т. д. Функция, представленная ниже, имеющая имя **int**_port(), используется для инициализации значений различных портов системы.

```
/* Инициализация порта */
void port_init(port, code)
int port;
unsigned char code;
{
  union REGS r;
  r.x.dx = port; /* последовательный порт */
  r.h.ah = 0;    /* функция инициализации порта */
  r.h.al = code; /* код инициализации - см. текст */
  int86(0x14, &r, &r);
}
```

Эта функция использует функцию int86(), поддерживаемую большинством компиляторов, включая Турбо Си и MicroSoft C. Если вы используете компилятор, где int86() не определена, то вместо нее может быть введено нечто (если пользователь сам не определил эту функцию), что может привести к ошибке. вы можете разработать свою специальную функцию инициализации последовательного порта. (Так в Турбо Си есть функция bioscom(), позволяющая инициализировать порт).

ПЕРЕДАЧА БАЙТОВ

Прерывание BIOS 14H, утилита 1 используется для передачи одного байта информации через последовательный порт, специфицированный содержимым регистра DX. Пересылаемый байт должен содержаться в регистре AL. Состояние процесса передачи возвращается в регистр AH. Функция sport() , представленная ниже, передает один байт из специфицированного последовательного порта.

```
/* Передача символа из последовательного порта */
void sport(port, c)
int port;                  /* порт ввода/вывода */
char c;                    /* передаваемый символ */
{
  union REGS r;
  r.x.dx = port;           /* последовательный порт */
```

```
  r.h.al = c;             /* передаваемый символ */
  r.h.ah = 1;             /* пересылка символа функции */
  int86(0x14, &r, &r);
  if(r.h.ah & 128) {      /* контроль 7-го бита */
    printf("обнаружена ошибка передачи в ");
    printf("последовательном порту");
    exit(1);
  }
}
```

Если бит 7 регистра AH получил значение после выполнения прерывания BIOS, то регистрируется ошибка передачи данных. Для определения причины ошибки вы должны считать состояние порта; как это сделать обсуждается ниже. Несмотря на то, что функция sport() при обнаружении ошибки прекращает свою работу, вы можете сохранить код ошибки в управляющей программе, а затем, определив тип ошибки, предусмотреть определенные действия по ее обработке.

КОНТРОЛЬ СОСТОЯНИЯ ПОРТА

Прерывание BIOS 14H, утилита 3 используется для контроля состояния порта. Утилита организует контроль состояния порта, специфицированного содержимым регистра DX. После возврата из состояния, определяемым прерыванием, регистры AH и AL будут содержать значения, определяющие в соответствии с Таблицей 6-3 текущее состояние порта после выполнения прерывания BIOS.

Таблица 6-3

Байты состояния последовательного порта

Состояние канала связи (AH)	
Значение, устанавливающее бит	Бит
Готовность данных	0
Ошибка переполнения	1
Ошибка контроля четности	2
Ошибка кодирования	3
Ошибка при идентификации прерывания	4
Регистр накопления передаваемых данных	5
Регистр сдвига передачи пуст	6
Выход за допустимый интервал времени	7
Состояние модема (AL)	
Значение, устанавливающее бит	Бит
Искажение в очистке-для-посылки	0
Искажение в наборе-данных-готов	1
Обнаружен задний фронт кольцевого импульса	2
Искажение сигнала в канале связи	3
Очистка-для-посылки	4
Набор-данных-готов	5
Признак кольца	6
Зафиксирован сигнал от канала связи	7

Как вы можете видеть, из многообразия различных состояний, анализируемых при использовании модема, в случае обеспечения связи последовательного порта с каким-либо иным устройством, используются лишь наиболее важные, а не весь представленный в Таблице 6-3 набор состояний. Однако, одно из состояний - "готовность данных" является чрезвычайно важным. Анализируя процесс передачи данных на возникновение этого состояния, вы можете определить, какие конкретно байты данных были получены портом и готовы для чтения. Функция rport() использует данные, считываемые ею с порта. На примере этой функции показано, каким образом используется возможность анализа состояния "готовность данных". Итак, перейдем к следующему разделу главы.

ПРИЕМ БАЙТОВ

Прерывание BIOS 14H, утилита 3 используется для чтения байтов из последовательного порта. Номер последовательного порта предварительно специфицируется содержимым регистра DX. После выхода из состояния, определяемого прерыванием BIOS, очередной символ считывается в регистр AL. После передачи символа и считывания его в регистр AL бит 7 регистра AH сигнализирует о результате выполнения операции получения-чтения символа (ошибка или норма).

Функция rport(), представленная ниже, выполняет чтение байта из специфицированного последовательного порта.

```
/* Чтение символа из порта */
rport(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  /* Ожидание прихода символа */
  while(!(check_stat(PORT)&256))
  if(kbhit()) { /* выход по прерыванию от клавиатуры */
    getch();
    exit(1);
  }
r.x.dx = port; /* последовательный порт */
r.h.ah = 2;    /* номер функции чтения */
int86(0x14, &r, &r);
if(r.h.ah & 128)
  printf("в последовательном порту обнаружена ошибка чтения");
return r.h.al;
}
```

Прерывание для чтения данных из порта не инициируется системой до тех пор, пока очередной байт не будет получен последовательным портом, и инициируется до того, как байт будет потерян регистром. Поэтому наиболее типичной ошибкой при чтении байта является отсутствие контакта с каналом связи, что приводит к зависанию компьютера. Для решения этой проблемы функция rport() анализирует состояние специфицированного порта, проверяя значение бита, индицирующего готовность данных. В то же время функция kbhit() контролирует поступление прерывания от клавиатуры. Если была нажата клавиша, то функция rport() прекращает свою работу. (вы можете предусмотреть в ряде случаев вызов какой-либо функции для обработки такой ситуации). Использование функции kbhit() позволяет получить возможность прекращения работы функции rport() в случае, если получение данных портом невозможно и, в свою очередь, предотвратить зависание компьютера. Как только данные получены, инициируется прерывание 14H, утилита 2, и очередной байт считывается функцией из порта, после чего анализируется бит

7 регистра AH на предмет результата выполнения операции (ошибка или норма). В конечном итоге, считанный байт возвращается функцией в вызывающую программу.

ПЕРЕДАЧА ФАЙЛОВ МЕЖДУ КОМПЬЮТЕРАМИ

Сегодня многие организации и частные лица имеют в своем распоряжении несколько компьютеров, причем часто эти компьютеры оказываются разных типов или разных моделей, а также имеют несовместимые форматы дисков. Например 3.5 дюймовые дискеты системы PS/2 несовместимы с 5.5 дюймовыми дискетами более ранних моделей компьютеров IBM - PC, XT, AT. При использовании различных компьютеров большое преимущество может быть достигнуто при соединении компьютеров через их последовательные порты с целью совместного использования ими информации и/или программ. Во многих случаях создание программ, обеспечивающих обмен файлами

для таких компьютеров через их последовательные порты, является проблематичным.

Однако существует довольно быстродействующая и эффективная программа передачи файлов. Эта программа подробно рассматривается в этой главе; она обладает рядом значительных преимуществ: она работает с любыми типами файлов на всех типах компьютеров, которые естественно отличаются друг от друга своей производительностью и, самое главное, не используют аппаратного подтверждения связи. Последняя особенность программы позволяет использовать трехжильный кабель. В добавок ко всему, программа может работать даже тогда, когда аппаратное подтверждение связи в принципе невозможно и бесполезно.

Но все равно вы можете использовать аппаратное подтверждение связи потому, что это позволяет достичь более высокого уровня производительности и надежности нежели организация взаимодействия компьютеров без него. Это связано с тем, что довольно часто генерация специальных сигналов программой затруднена и программно реализованные сигналы часто претерпевают искажения, а также зачастую бесполезны вообще. Эта ситуация (при объединении компьютеров) будет существовать еще очевидно довольно долго, являясь в то же время достаточно общей.

Подпрограммы передачи файлов выполняют свои функции, используя программное подтверждение связи, и функционируют фактически в различных средах. Однако для решения глобальной проблемы лучше пожертвовать производительностью, увеличив надежность системы.

ПРОГРАММНОЕ ПОДТВЕРЖДЕНИЕ СВЯЗИ

Когда аппаратное подтверждение связи невозможно или бесполезно, единственным способом, позволяющим избежать ошибок переполнения регистра, которые не могут быть зарегистрированы непосредственно во время передачи данных по каналу связи, является введение программного подтверждения связи. Программное подтверждение связи работает следующим образом: компьютер-источник посылает первый байт и переходит в состояние ожидания возврата от компьютера-приемника квитирующего байта (байта, подтверждающего принятие предыдущего сообщения). При получении квитирующего байта компьютер-источник посылает следующий байт и снова переходит в состояние ожидания квитирующего байта от компьютера-приемника.

Этот процесс продолжается до тех пор, пока весь файл целиком не будет передан. Ниже представлены в терминах псевдо-Си процедуры передачи и приема данных.

```
send()
{
   while ( есть байты для передачи ){
      send( байт );
      wait();
   }
}
receive()
{
   do {
      receive_byte();
      send( квитирующй байт );
   } while( пока все байты не считаны );
}
```

При этом подходе передача данных не вызовет никогда переполнения регистра в порте-приемнике независимо от того, насколько велика разница в скорости выполнения операций компьютеров, между которыми установлена связь.

При этом типе подтверждения связи имеется лишь один недостаток - скорость передачи данных падает вдвое по сравнению с

теоретически возможной. Это объясняется тем, что при передаче одного байта информации фактически происходит передача двух байт (вспомните о квитирующем байте).

СЕМЬ ИЛИ ВОСЕМЬ БИТ ДАННЫХ

Если вы собираетесь организовать передачу только текстовых файлов, то вы вполне можете использовать лишь семь бит под данные по той лишь причине, что ни одна буква или символ пунктуации не требует для своего представления восемь бит. Передавая только семь бит, вы даже незначительно увеличите скорость передачи файла. Но как быть, если необходимо передать не текстовый файл, а программу?

Все файлы, содержащие программы (выполняемые) и некоторые виды файлов данных, используют восьмибитовое представление данных, то есть весь байт. По этой причине для передачи файла, содержащего выполняемую программу, программа передачи файлов должна передавать все восемь бит. Однако существует еще одна проблема, возникающая при передаче двоичных файлов: EOF (символ End-Of-File) не используется для сигнализации об окончании файла. Для решения этой проблемы число байтов в файле должно быть передано порту-приемнику до передачи всего файла.

ПЕРЕКАЧКА ФАЙЛА

Первой необходимой нам подпрограммой является функция, обеспечивающая передачу файла через последовательный порт. В общем случае эта функция должна открыть файл, который будет передан в другой компьютер, подсчитать его длину, передать в порт -приемник длину передаваемого файла и, в конце концов, перекачать сам файл. Функция send_file(), представленная ниже, как раз и предназначена для решения этих задач.

```
/* перекачка специфицированного файла */
void send_file(fname)
char *fname;
{
  FILE *fp;
  char ch;
  union {
    char c[2];
    unsigned int count;
  } cnt;
  if(!(fp=fopen(fname,"rb"))) {
    printf("Входной файл не может быть открыт\n");
    exit(1);
  }
  send_file_name(fname);  /* передача имени файла */
  wait(PORT);  /* ожидание квитирующего байта */
  /* вычисление размера выходного файла */
  cnt.count = filesize(fp);
  /* размер посылки */
  sport(PORT, cnt.c[0]);
  wait(PORT);
  sport(PORT, cnt.c[1]);
  do {
    ch = getc(fp);
    if(ferror(fp)) {
      printf(" ошибка чтения выходного файла\n");
      break;
    }
    /* ожидание готовности порта-приемника */
    if(!feof(fp)) {
      wait(PORT);
      sport(PORT, ch);
```

```
    }
  } while(!feof(fp));
  wait(PORT);/* ожидание подтверждения получения последнего байта
*/

  fclose(fp);
}
```

Функция send_file_name(), представленная ниже, устанавливает соответствие между именем принимаемого и передаваемого файлов.

```
/* Перекачка имени файла */
void send_file_name(f)
char *f;
{
  printf(" Ожидание передачи... \n");
  do {
    sport(PORT, '?');
  } while(!kbhit() && !(check_stat(PORT)&256));
  if(kbhit()) {
    getch();
    exit(1);
  }
  wait(PORT); /* ожидание получения квитирующего байта */
  printf("Передано %s\n\n",f);
  /* фактическая передача имени файла */
  while(*f) {
    sport(PORT, *f++);
    wait(PORT); /* ожидание получения квитирующего байта  */
  }
  sport(PORT,'\0'); /* символ конца строки */
}
```

Функция send_file_name() предназначена для решения двух основных задач. Во-первых, она устанавливает связь с компьютером-приемником путем передачи ему маркера вопроса ('?') и дожидается ответа от него в виде квитирующего байта. (В качестве квитирующего символа используется точка. Однако вы можете по своему усмотрению использовать другой символ. После того, как связь будет установлена, осуществляется передача имени файла. Заметьте, что эта функция завершает аварийно свою работу при поступлении прерывания от клавиатуры.

Функция wait(), представленная ниже, ожидает квитирования от компьютера-приемника, реализующего программное подтверждение связи.

```
/* ожидание ответа */
void wait(port)
int port;
{
  if(rport(port)!='.') {
    printf("ошибка установления связи \n");
    exit(1);

  }
}
```

Таким образом, при обнаружении ошибки эта функция прекращает свою работу. Однако вы можете предусмотреть обработку данной ситуации.

Функция filesize() возвращает размер файла в байтах. Ее использование возможно, если ваш компилятор Си поддерживает функцию вычисления длины файла, в противном случае вы должны заменить эту функцию разработанной вами, но выполняющей аналогичные действия. Переменная cnt, входящая в состав структуры **union**, служит для хранения двухбайтовой длины файла, но вы должны помнить, что за единицу времени вы можете переслать через последовательный порт только один байт.

ПРИЕМ ФАЙЛА

Прием файла является прямо противоположной операцией передачи файла. Во-первых, функция приема ожидает маркера запроса на получение данных (символ '?'). На получение маркера функция отвечает точкой (символом квитирования). После получения имени файла функция ожидает получение его размера в байтах. В конечном итоге функция начинает чтение файла. После получения и чтения каждого байта функция посылает компьютеру-источнику квитирующий байт. Таким образом она реализует программное подтверждение связи. Функция rec_file() представлена ниже.

```
                        /* Прием файла */
void rec_file()
{
  FILE *fp;
  char ch;
  char fname[14];
  union {
    char c[2];
    unsigned int count;
  } cnt;
  get_file_name(fname); /* получение имени файла */
  printf(" Получен файл %s\n",fname);
  remove(fname);
  if(!(fp=fopen(fname, "wb"))) {
    printf(" Невозможно открыть выходной файл \n");
    exit(1);
  }
  /* Получение длины файла */
  sport(PORT, '.'); /* квитирование */
  cnt.c[0] = rport(PORT);
  sport(PORT, '.'); /* квитирование */
  cnt.c[1] = rport(PORT);
  sport(PORT, '.'); /* квитирование */
  for(; cnt.count; cnt.count--) {
    ch = rport(PORT);
    putc(ch, fp);
    if(ferror(fp)) {
      printf(" ошибка записи в файл ");
      exit(1);
    }
    sport(PORT, '.'); /* квитирование */
  }
  fclose(fp);
}
```

Функция get_file_name() представлена ниже.

```
/* Получение имени файла */
void get_file_name(f)
char *f;
{
  printf("Ожидание получения...\n");
  while(rport(PORT)!='?') ;
  sport(PORT, '.'); /* квитирование */
  while((*f=rport(PORT))) {
    if(*f!='?') {
      f++;
      sport(PORT, '.'); /* квитирование */
    }
   }
 }
```

ПЕРЕКАЧКА ПРОГРАММЫ

Файл, который обеспечивает перекачку программы из компьютера в компьютер, включающий все необходимые функции поддержки, представлен в данном параграфе. Программа перекачки использует последовательный порт с именем 0 - первый последовательный порт; однако, изменяя значения макроопределения PORT в начале текста программы, вы можете использовать другие порты.

```
/*   Программа перекачки файла, использующая
     программное подтверждение связи.
     Порт инициализирован с параметрами:
          скорость передачи - 9600 бод,
          контроль четности/нечетности не производится,
          восемь бит данных,
          два завершающих стоп-бита.
*/
#define PORT 0
#include "dos.h"
#include "stdio.h"
unsigned int filesize();
void sport(), send_file(), rec_file(), send_file_name();
void get_file_name(), port_init(), wait();
main(argc,argv)
int argc;
char *argv[];
{
  if(argc<2) {
 printf(" Используйте формат TRANS S <имя файла> или TRANS R\n");
    exit(1);
  }
  printf("Задача перекачки программ запущена. Для аварийного\n");
  printf("завершения нажмите любую клавишу.\n\n");
  port_init(PORT, 231); /* инициализация последовательного порта
                        */
  if(tolower(*argv[1]) == 's') send_file(argv[2]);
  else rec_file();
}
/* перекачка специфицированного файла */
void send_file(fname)
char *fname;
{
  FILE *fp;
  char ch;
  union {

    char c[2];
    unsigned int count;
  } cnt;

  if(!(fp=fopen(fname,"rb"))) {
    printf("Входной файл не может быть открыт\n");
    exit(1);
  }
  send_file_name(fname);  /* передача имени файла */
  wait(PORT);  /* ожидание квитирующего байта */
  /* вычисление размера выходного файла */
  cnt.count = filesize(fp);
  /* размер посылки */
  sport(PORT, cnt.c[0]);
  wait(PORT);
  sport(PORT, cnt.c[1]);
  do {
    ch = getc(fp);
    if(ferror(fp)) {
      printf(" ошибка чтения выходного файла\n ");
      break;
```

```
    }
    /* ожидание готовности порта-приемника */
    if(!feof(fp)) {
      wait(PORT);
      sport(PORT, ch);
    }
  } while(!feof(fp));
  wait(PORT);/* ожидание подтверждения получения последнего байта
*/
  fclose(fp);
}
/* прием файла */
void rec_file()
{
  FILE *fp;
  char ch;
  char fname[14];
  union {
    char c[2];
    unsigned int count;
  } cnt;
  get_file_name(fname); /* получение имени файла */
  printf("Получен файл %s\n",fname);
  remove(fname);
  if(!(fp=fopen(fname, "wb"))) {

    printf(" Невозможно открыть выходной файл \n");
    exit(1);
  }
  /* Получение длины файла */
  sport(PORT, '.'); /* квитирование */
  cnt.c[0] = rport(PORT);
  sport(PORT, '.'); /* квитирование */
  cnt.c[1] = rport(PORT);
  sport(PORT, '.'); /* квитирование */
  for(; cnt.count; cnt.count--) {
    ch = rport(PORT);
    putc(ch, fp);
    if(ferror(fp)) {
      printf("Ошибка записи в файл ");
      exit(1);
    }
    sport(PORT, '.'); /* квитирование */
  }
  fclose(fp);
}
/* Возвращение значения длины файла в байтах */
unsigned int filesize(fp)
FILE *fp;
{
  unsigned long int i;
  i = 0;
  do {
    getc(fp);
    i++;
  } while(!feof(fp));
  rewind(fp);
  return (i-1); /* Не считая символ EOF */
}
/* Перекачка имени файла */
void send_file_name(f)
char *f;
{
  printf(" ожидание передачи... \n");
```

```
  do {
    sport(PORT, '?');
  } while(!kbhit() && !(check_stat(PORT)&256));
  if(kbhit()) {
    getch();
    exit(1);
  }
  wait(PORT); /* ожидание получения квитирующего байта */
  printf("Передано %s\n\n",f);
  /* фактическая передача имени файла */

  while(*f) {
    sport(PORT, *f++);
    wait(PORT); /* ожидание получения квитирующего байта  */
  }
  sport(PORT, '\0'); /* символ конца строки */
}
/* Получение имени файла */
void get_file_name(f)
char *f;
{
  printf(" ожидание получения...\n");
  while(rport(PORT)!='?');
  sport(PORT, '.');     /* квитирование */
  while((*f=rport(PORT))) {
    if(*f!='?') {
      f++;
      sport(PORT, '.'); /* квитирование */
    }
   }
 }
/* Ожидание ответа */
void wait(port)
int port;
{
  if(rport(port)!='.') {
    printf("ошибка установления связи \n");
    exit(1);
  }
}
/* Передача символа из последовательного порта */
void sport(port, c)
int port; /* порт ввода/вывода */
char c;   /* пересылаемый символ */
{
  union REGS r;
  r.x.dx = port; /* последовательный порт */
  r.h.al = c; /* символ для передачи */
  r.h.ah = 1; /* функция передачи символа */
  int86(0x14, &r, &r);
  if(r.h.ah & 128) {
  printf("ошибка при передаче данных в последовательном порту ");
    exit(1);
  }
}
/* чтение символа из последовательного порта */
rport(port)
int port; /* порт ввода/вывода */
{
  union REGS r;

  /* ожидание символа */
  while(!(check_stat(PORT)&256))
    if(kbhit()) { /* аварийное завершение по прерыванию с
```

```
                        клавиатуры */
       getch();
       exit(1);
     }
  r.x.dx = port; /* последовательный порт */
  r.h.ah = 2; /* функция чтения символа  */
  int86(0x14, &r, &r);
  if(r.h.ah & 128)
   printf(" обнаружена ошибка чтения в последовательном порту ");
  return r.h.al;
}
/* контроль состояния последовательного порта */
cheek_stat(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  r.x.dx = port; /* последовательный порт  */
  r.h.ah = 3;    /* чтение состояния */
  int86(0x14, &r, &r);
  return r.x.ax;
}
/* инициализация порта
*/
void port_init(port, code)
int port;
unsigned char code;
{
  union REGS r;
  r.x.dx = port; /* последовательный порт */
  r.h.ah = 0;    /* функция инициализации порта*/
  r.h.al = code; /* код инициализации - см. выше */
  int86(0x14, &r, &r);
}
```

ИСПОЛЬЗОВАНИЕ СРЕДСТВ ПЕРЕКАЧКИ ПРОГРАММ

Программа перекачки обрабатывает данные в соответствии с параметрами в командной строке. Во всех случаях программа перекачки вызывается по имени TRANS . Она выполняет передачу файла, используя следующие основные формы вызова:

TRANS S <имя_файла>,

где <имя_файла> - имя файла, который требуется передать в другой компьютер через последовательный порт.

Для получения файла необходимо выдать команду:

TRANS R

При получении файла специфицировать его имя нет необходимости в связи с тем, что имя передаваемого файла посылается перед его непосредственной передачей из компьютера - источника.

ДАЛЬНЕЙШЕЕ СОВЕРШЕНСТВОВАНИЕ ПРОГРАММЫ

Программа перекачки файлов является функционально полной, совершенно безопасной и надежной. Естественно, что при эксплуатациии программы вам может встретиться ряд критических ситуаций, для которых даже не установлены соответствующие коды ошибок. В этом случае вы, возможно, захотите несколько усовершенствовать эту программу, добавив в нее новые функции.

Одним из путей выявления критических ситуаций при передаче данных является обеспечение режима "эхо" для каждого полученного байта, реализуемого путем использования в качестве квитирующего байта только что полученного байта информации. Для этого надо доработать функцию передачи. Она, в частности, должна будет проводить сравнение переданного байта с соответствующим этой

передаче квитирующим байтом. При обнаружении различий этих байтов функция должна информировать об ошибке.

Можно также доработать программу так, чтобы она осуществляла попытку повторить действия, вызывающие ошибку, а не прекращала функционирование при обнаружении ошибки. Следует отметить, что автоматический перезапуск функций в программе перекачки файлов значительно усложняет как функции передачи, так и функции получения файлов. Но в то же время затраты полностью окупятся тем, что выполнение программы на одном, а может быть сразу и на двух компьютерах сможет в этом случае обойтись без непосредственного сопровождения пользователем.

И, наконец, вам может понадобиться выдача причины возникновения той или иной ошибки в процессе передачи файлов. Это свойство программы очень поможет вам при решении проблем диагностики процесса передачи файлов из компьютера в компьютер.

ПРОСТЕЙШАЯ ЛВС

Локальные вычислительные сети (ЛВС) получают все большую популярность при совместном использовании множества компьютеров. Эти сети обеспечивают передачу как данных, так и программ между множеством различных компьютеров. Существует два основных способа объединения компьютеров в ЛВС. Первый метод состоит в объединении всех компьютеров в сеть, причем любой компьютер может обратиться за информацией или программой к любому другому компьютеру. Такой способ объединения называется сетью с кольцевой топологией. Однако, этот тип сетей кроме всех его преимуществ обладает тремя крупными недостатками, которые обуславливают довольно редкое его использование. Во-первых, это трудность (хотя эта проблема и разрешима) обеспечения безопасности информации. Во-вторых, управление данными и программами должно выполняться комплексно, так как централизованного размещения определенных файлов добиться невозможно. В-третьих, каждый компьютер, включенный в сеть, должен постоянно выделять часть своих вычислительных ресурсов на пересылку различных файлов пользователей, что значительно понижает производительность каждого компьютера.

Вторым, более общим методом создания ЛВС является сеть звездообразной топологии. Этот метод использует центральный компьютер-диспетчер для хранения файлов и обеспечения ими других компьютеров сети. Центральный компьютер часто называют файловым сервером (file server). Компьютеры, имеющие доступ к файловому серверу, в зависимости от производительности и специфики использования называются узлами сети (nodes), терминалами (terminals) или рабочими станциями (workstations).

Особенности топологии двух типов сетей иллюстрирует рисунок 6-1. В данном параграфе рассматривается сеть звездообразной топологии. В действительности в заголовке параграфа есть преувеличение. В настоящих ЛВС файловый сервер "прозрачен" для всех абонентов сети и лишь расширяет возможности рабочих станций ЛВС по непосредственному доступу к файлам файлового сервера. Программы, представленные в этом параграфе, используются рабочей станцией ЛВС для явного указания файла и доступа к нему. Таким образом, этот подход облегчает дальнейшее развитие программного обеспечения, так как не требует специальных аппаратных средств для реализации файлового сервера. Вы можете использовать эти программы в качестве стартовой точки при разработке всего программного обеспечения ЛВС.

ФАЙЛОВЫЙ СЕРВЕР

Файловый сервер находится в центре сети звездообразной топологии и осуществляет последовательный контроль состояний каждого последовательного порта в системе. Рабочая станция сигнализирует о требовании на получение или передачу файла,

помещая символ "r" или "s" в свой порт. Символ "s" означает требование на передачу файла; символ "r" означает требование на получение файла (и сохранение его) с помощью файлового сервера.

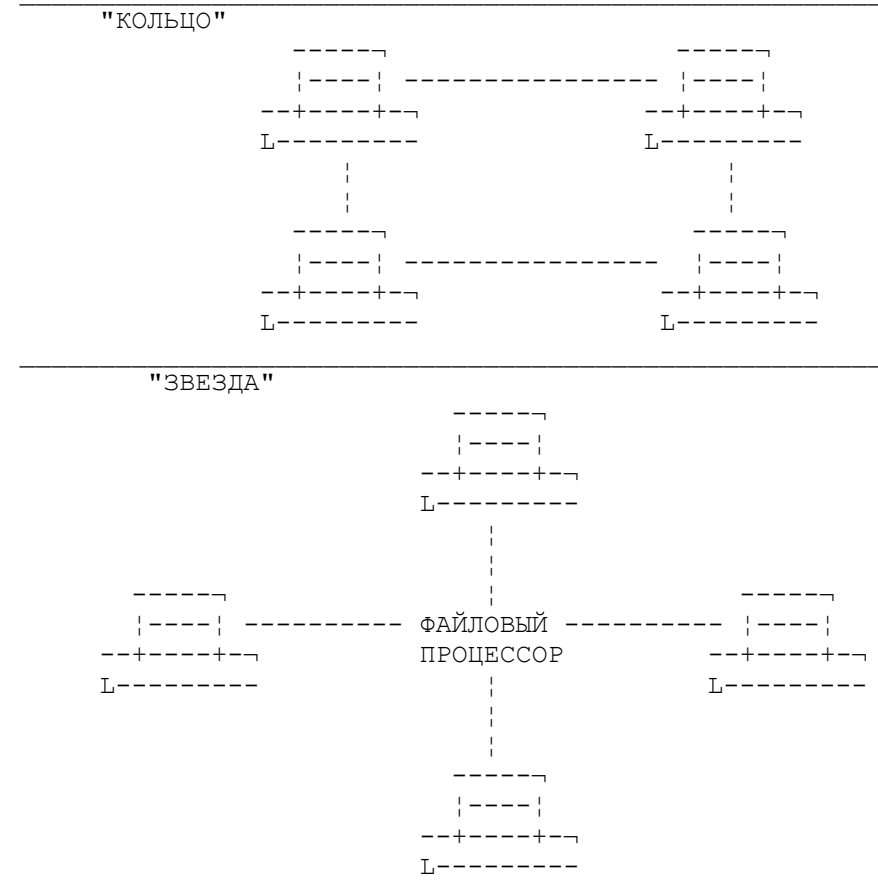


Рис. 6.1. Сети кольцевой и звездообразной топологии.

При регистрации появления в одном из портов маркера, соответствующего требованию на получение или передачу данных, файловый сервер выполняет его, а затем осуществляет последовательный контроль состояний всех последовательных портов в ожидании нового запроса на пересылку файлов. В действительности получение или передача файла в сети базируется на использовании программы перекачки файлов из первой части главы.

Основной цикл работы файлового сервера представлен ниже. Тексты программ, вставленные в виде комментария, позволяют проследить основной цикл работы файлового сервера при подключении к нему новых портов (новых абонентов в сеть).

```
main()
{
  printf("Работает файловый сервер./n");
  printf("Для выхода нажмите любую клавишу./n/n");
  port_init(PORT); /* инициализации последовательного порта */
  do {
    /*ожидание запроса на обработку файла  */
    if(check_stat(PORT)&256) {
      switch(rport(PORT)) {
        case 's': send_file(PORT);
          break;
        case 'r': rec_file(PORT);
          break;
      }
    }
/************************************
При подключении новых рабочих станций
```

```
контроль состояния дополнительных портов
как приведено ниже...
    if(check_stat(PORT1)&256) {
      switch(rport(PORT1)) {
        case 's': send_file(PORT1);
          break;
        case 'r': rec_file(PORT1);
          break;
      }
    }
.
.
.
    if(check_stat(PORTn)&256) {
      switch(rport(PORTn)) {
        case 's': send_file(PORTn);
          break;
        case 'r': rec_file(PORTn);
          break;

      }
    }
*****************************************/
  } while(!kbhit());
}
```

Как видите, файловый сервер работает только с одной рабочей станцией (абонентом сети), однако, как указано в комментарии, он может работать в принципе с N абонентами сети. Заметьте, что файловый сервер работает до тех пор, пока не поступило прерываний с клавиатуры. Это позволяет ему всегда быть в состоянии готовности обработки очередного требования на передачу/получение файла.

Как вы можете видеть, функции send_file() и rec_file() теперь осуществляют обработку порта, который передается им как аргумент. Это объясняется необходимостью обработки файловым сервером множества различных последовательных портов. В функции файлового сервера входит также передача квитирующего символа абонентам в случае получения от них требования на передачу файла в файловый сервер. Модификация функций send_file() и rec_file() для работы в файловом сервере приведена ниже.

```
/* Перекачка специфицированного файла через последовательный порт
*/
void send_file(port)
int port;
{
  FILE *fp;
  char ch, fname[14];
  union {
    char c[2];
    unsigned int count;
  } cnt;
  sport(port, '.'); /* квитирование */
  get_file_name(fname, PORT);
  if(!(fp=fopen(fname,"rb"))) {
    printf("Входной файл не может быть открыт\n");
    exit(1);
  }
  if(rport(port)!='.') {
     printf("Сбой при работе с удаленным файлом\n");
     exit(1);
  }
  printf("Пересылается файл %s\n", fname);
  /* Определение размера файла */
  cnt.count = filesize(fp);
```

```
  /* Передача размера файла */


  sport(port, cnt.c[0]);
  wait(port);
  sport(port, cnt.c[1]);
  do {
    ch = getc(fp);
    if(ferror(fp)) {
       printf("Ошибка чтения входного файла\n");
       break;
    }
    /*Ожидание готовности получателя*/
    if(!feof(fp)) {
      wait(port);
      sport(port, ch);
    }
  } while(!feof(fp));
  wait(port); /*чтение последней порции данных из порта*/
  fclose(fp);
}
/*Получение файла через последовательный порт*/
void rec_file(port)
int port;
{
  FILE *fp;
  char ch, fname[14];
  union {
    char c[2];
    unsigned int count;
  } cnt;
  sport(port, '.'); /* квитирование */
  get_file_name(fname, PORT);
  printf("Получен файл %s\n", fname);
  remove(fname);
  if(!(fp=fopen(fname,"wb"))) {
    printf("Выходной файл не может быть открыт\n");
    exit(1);
  }
  /*считывание длины файла*/
  sport(port, '.');
  cnt.c[0] = rport(port);
  sport(port, '.');
  cnt.c[1] = rport(port);
  sport(port, '.');
  for(; cnt.count; cnt.count--) {
    ch = rport(port);
    putc(ch, fp);
    if(ferror(fp)) {


      printf("Ошибка при записи файла\n");
      exit(1);
    }
    sport(port, '.');
  }
  fclose(fp);
}
```

Полностью программа, реализующая файловый сервер, приведена ниже. Эта программа использует порт с именем 0. Однако, если вы имеете более одного абонента в сети, то вы должны добавить в эту программу соответствующие операторы (см. основной рабочий цикл файлового сервера) для обработки порта нового абонента.

```
/* Простейший файловый сервер ЛВС. Параметры порта:
```

```
          скорость передачи - 9600 бод,
          контроль четности       выкл. ,
          восемь бит данных,
          два завершающих стоп-бита.
*/
#define PORT 0
#include "dos.h"
#include "stdio.h"
unsigned int filesize();
void sport(), send_file(), rec_file(), send_file_name();
void get_file_name(), port_init(), wait();
main()
{
  printf("Работает файловый сервер.\n");
  printf("Для выхода нажмите любую клавишу./n/n");
  port_init(PORT); /* инициализации последовательного порта */

  do {
   /*ожидание запроса на обработку файла*/
   if(check_stat(PORT)&256) {
     switch(rport(PORT)) {
        case 's': send_file(PORT);
          break;
        case 'r': rec_file(PORT);
          break;
      }
    }
/****************************************
При подключении новых рабочих станций
контроль состояния дополн. портов, как


приведено ниже...
   if(check_stat(PORT1)&256) {
     switch(rport(PORT1)) {
        case 's': send_file(PORT1);
          break;
        case 'r': rec_file(PORT1);
          break;
      }
    }

   if(check_stat(PORTn)&256) {
     switch(rport(PORTn)) {
        case 's': send_file(PORTn);
          break;
        case 'r': rec_file(PORTn);
          break;
      }
    }
****************************************/
  } while(!kbhit());
}
/* Перекачка специфицированного файла через последовательный порт
*/
void send_file(port)
int port;
{
  FILE *fp;
  char ch, fname[14];
  union {
    char c[2];
    unsigned int count;
  } cnt;
```

```
  sport(port, '.'); /* квитирование */
  get_file_name(fname, PORT);
  if(!(fp=fopen(fname,"rb"))) {
    printf("Входной файл не может быть открыт\n");
    exit(1);
  }
  if(rport(port)!='.') {
     printf("Сбой при работе с удаленным файлом\n");
     exit(1);
  }
  printf("Пересылается файл %s\n", fname);
  /* Определение размера файла */
  cnt.count = filesize(fp);
  /* Передача размера файла */


  sport(port, cnt.c[0]);
  wait(port);
  sport(port, cnt.c[1]);
  do {
    ch = getc(fp);
    if(ferror(fp)) {
       printf("Ошибка чтения входного файла\n");
       break;
    }
    /*Ожидание готовности получателя*/
    if(!feof(fp)) {
      wait(port);
      sport(port, ch);
    }
  } while(!feof(fp));
  wait(port); /*чтение последней порции данных из порта*/
  fclose(fp);
}
/*Передача специфицированного файла через последовательный
порт.*/
void rec_file(port)
int port;
{
  FILE *fp;
  char ch, fname[14];
  union {
    char c[2];
    unsigned int count;
  } cnt;
  sport(port, '.'); /* квитирование */
  get_file_name(fname, PORT);
  printf("Получен файл %s\n", fname);
  remove(fname);
  if(!(fp=fopen(fname,"wb"))) {
    printf("Выходной файл не может быть открыт\n");
    exit(1);
  }
  /*считывание длины файла*/
  sport(port, '.');
  cnt.c[0] = rport(port);
  sport(port, '.');
  cnt.c[1] = rport(port);
  sport(port, '.');
  for(; cnt.count; cnt.count--) {
    ch = rport(port);
    putc(ch, fp);
```

```
    if(ferror(fp)) {
      printf("Ошибка при записи файла\n");
      exit(1);
    }
    sport(port, '.');
  }
  fclose(fp);
}
/* Возвращение значения длины файла в байтах */
unsigned int filesize(fp)
FILE *fp;
{
  unsigned long int i;
  i = 0;
  do {
    getc(fp);
    i++;
  } while(!feof(fp));
  rewind(fp);
  return (i-1); /* Не считая символ EOF */
}
/* Перекачка имени файла */
void send_file_name(f, port)
char *f;
int port;
{
  do {
    sport(port, '?');
  } while(!kbhit() && !(check_stat(port)&256));
  if(kbhit()) {
    getch();
    exit(1);
  }
  wait(port);
  while(*f) {
    sport(port, *f++);
    wait(port); /* ожидание получения квитирующего байта  */
  }
  sport(port, 0); /* символ конца строки */
}
/* Получение имени файла */
void get_file_name(f, port)
char *f;
int port;
{
  while(rport(port)!='?') printf(".");
  sport(port, '.');


  while((*f=rport(port))) {
    if(*f!='?') {
      f++;
      sport(port, '.');
    }
   }
   sport(port, '.');
 }
/* ожидание ответа */
void wait(port)
int port;
{
  if(rport(port)!='.') {
    printf("ошибка установления связи \n");
    exit(1);
```

```
  }
}
/* Передача символа из последовательного порта */
void sport(port, c)
int port;                  /* порт ввода/вывода */
char c;                    /* передаваемый символ */
{
  union REGS r;
  r.x.dx = port;           /* последовательный порт */
  r.h.al = c;              /* передаваемый символ */
  r.h.ah = 1;              /* пересылка символа функции */
  int86(0x14, &r, &r);
  if(r.h.ah & 128) {       /* контроль 7-го бита */
  printf("Обнаружена ошибка передачи в последовательном порту ");
  printf("%d",r.h.ah);
    exit(1);
  }
}
/* Чтение символа из порта */
rport(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  /* Ожидание прихода символа */
while(!(check_stat(port)&256))
  if(kbhit()) { /* выход по прерыванию от клавиатуры */
    getch();
    exit(1);
  }
r.x.dx = port; /* последовательный порт */
r.h.ah = 2;    /* функция чтения символа */
int86(0x14, &r, &r);
if(r.h.ah & 128)
  printf("В последовательном порту обнаружена ошибка чтения");


return r.h.al;
}
/* Проверка состояния последовательного порта */
check_stat(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  r.x.dx = port; /* последовательный порт  */
  r.h.ah = 3;    /* чтение состояния */
  int86(0x14, &r, &r);
  return r.x.ax;
}
/* инициализация порта с параметрами:
скорость передачи 9600 бод, два стоп-бита,
контроль на четность  выкл., 8 бит данных.
*/
void port_init(port)
int port;
{
  union REGS r;
  r.x.dx = port; /* последовательный порт */
  r.h.ah = 0;    /* функция инициализации порта*/
  r.h.al = 231; /* код инициализации - см. выше */
  int86(0x14, &r, &r);
}
```

Для того, чтобы рабочая станция инициировала требования на получение файла из файлового сервера и его загрузку, требуется вызов специальной программы. Эта программа вызывается по имени GET и выполняется рабочей станцией, которая нуждается в данных. Вы можете организовать вызов этой программы как команды расширенного набора команд DOS. Основной формой вызова программы GET является следующий:

GET <имя_файла>

где <имя_файла> - имя загружаемого файла.

Процесс функционирования функции GET имеет два отличия от процесса функционирования других функций, использующих файловый сервер.

Во-первых функция rec_file() пересылает имя файла компьютеру -получателю.

Во-вторых, имя порта жестко кодируется в подпрограммах, а не передается подпрограммам в качестве аргумента, как это делается в файловом сервере.

Полный текст программы GET представлен ниже.

```
/* Загрузка файла из файлового сервера. */
#define PORT 0
#include "dos.h"
#include "stdio.h"
void sport(), send_file(), rec_file(), send_file_name();
void get_file_name(), port_init(), wait();
main(argc,argv)
int argc;
char *argv[];
{
  if(argc!=2) {
    printf(" Используйте формат: GET <имя файла>\n");
    exit(1);
  }
  port_init(PORT); /* инициализация последовательного порта */
    rec_file(argv[1]);
  }
  /*Получение файла*/
void rec_file(fname)


char *fname;
{
  FILE *fp;
  char ch;
  union {
    char c[2];
    unsigned int count;
  } cnt;
  printf("Загружается файл %s\n", fname);
  remove(fname);
  if(!(fp=fopen(fname,"wb"))) {
    printf("Выходной файл не может быть открыт\n");
    exit(1);
  }
  sport(PORT, 's');     /*Передача серверу маркера
                          "готов к приему файла"*/
  wait(PORT);          /* Ожидание готовности сервера */
  /* Получение длины файла */
  send_file_name(fname);
  sport(PORT, '.'); /* квитирование */
  cnt.c[0] = rport(PORT);
  sport(PORT, '.'); /* квитирование */
  cnt.c[1] = rport(PORT);
  sport(PORT, '.'); /* квитирование */
```

```
  for(; cnt.count; cnt.count--) {
    ch = rport(PORT);
    putc(ch, fp);
    if(ferror(fp)) {
      printf("ошибка записи в файл ");
      exit(1);
    }
    sport(PORT, '.'); /* квитирование */
  }
  fclose(fp);
}
/* Перекачка имени файла */
void send_file_name(f)
char *f;
{
  do {
    sport(PORT, '?');
  } while(!kbhit() && !(check_stat(PORT)&256));
  if(kbhit()) {
    getch();
    exit(1);
  }
  wait(PORT);


  while(*f) {
    sport(PORT, *f++);
    wait(PORT);
  }
  sport(PORT, '\0'); /* символ конца строки */
  wait(PORT);
}
/*Ожидание ответа (квитирования)*/
void wait(port)
int port;
{
  if(rport(port)!='.') {
    printf("ошибка установления связи \n");
    exit(1);
  }
}

/* Передача символа из последовательного порта */
void sport(port, c)
int port;                  /* порт ввода/вывода */
char c;                    /* передаваемый символ */
{
  union REGS r;
  r.x.dx = port;           /* последовательный порт */
  r.h.al = c;              /* передаваемый символ */
  r.h.ah = 1;              /* пересылка символа функции */
  int86(0x14, &r, &r);
  if(r.h.ah & 128) {       /* контроль 7-го бита */
  printf("Обнаружена ошибка передачи в последовательном порту ");
  printf("%d",r.h.ah);
    exit(1);
  }
}
/* Чтение символа из порта */
rport(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  /* Ожидание прихода символа */
```

```
while(!(check_stat(port)&256))
  if(kbhit()) {
    getch();
    exit(1);
  }
r.x.dx = port; /* последовательный порт */
r.h.ah = 2;    /* функция чтения символа */
int86(0x14, &r, &r);
if(r.h.ah & 128)
  printf("в последовательном порту обнаружена ошибка чтения");
return r.h.al;


}
/* Проверка состояния последовательного порта */
check_stat(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  r.x.dx = port; /* последовательный порт  */
  r.h.ah = 3;    /* чтение состояния */
  int86(0x14, &r, &r);
  return r.x.ax;
}
/* инициализация порта с параметрами:
скорость передачи 9600 бод, два стоп-бита,
контроль на четность  выкл., 8 бит данных.
*/
void port_init(port)
int port;
{
  union REGS r;
  r.x.dx = port; /* последовательный порт */
  r.h.ah = 0;    /* функция инициализации порта*/
  r.h.al = 231; /* код инициализации - см. выше */
  int86(0x14, &r, &r);
}
```

ХРАНЕНИЕ ФАЙЛОВ

В большинстве сетей файлы могут не только пересылаться в узел сети от файлового сервера для обработки, но и пересылаться в обратном порядке - от абонента сети в сервер для хранения. Для поддержки этих возможностей была разработана программа PUT. Программа PUT выполняется в узле сети на рабочей станции и осуществляет перекачку файлов из узла сети в файловый сервер. Использование этой программы аналогично использованию программы GET (за исключением того, что выполняемые ими функции прямо противоположны). Вот основной формат вызова программы:

PUT <имя_файла>

Процесс выполнения программы PUT совершенно идентичен процессу выполнения программы, решающей задачу перекачки программных файлов.

Полный текст программы PUT приведен ниже.

```
#define PORT 0
#include "dos.h"
#include "stdio.h"
unsigned int filesize();
void sport(), send_file(), send_file_name();
void wait(), port_init(), wait();
main(argc,argv)
int argc;
char *argv[];
```

```
{
  if(argc!=2) {
    printf(" Используйте формат GET <имя файла>\n");
    exit(1);
  }
  port_init(PORT); /* инициализация последовательного порта */
  send_file(argv[1]);
  }
/* перекачка специфицированного файла */
void send_file(fname)
char *fname;
{
  FILE *fp;
  char ch;
  union {
    char c[2];
    unsigned int count;


  } cnt;
  if(!(fp=fopen(fname,"rb"))) {
    printf("Входной файл не может быть открыт\n");
    exit(1);
  }
  printf("Пересылается файл %s\n", fname);
/* Требуется файловый сервер.*/
  sport(PORT, 'r'); /* маркер готовности к пересылке файла
                       из узла */
  wait(PORT);/*ожидание готовности файлового сервера.*/
  send_file_name(fname);  /* передача имени файла */
  if(rport(PORT)!='.') {
    printf("Сбой при работе с удаленным файлом\n");
    exit(1);
  }
 /* вычисление размера выходного файла */
  cnt.count = filesize(fp);
  /* передача размера файла*/
  sport(PORT, cnt.c[0]);
  wait(PORT);
  sport(PORT, cnt.c[1]);
  do {
    ch = getc(fp);
    if(ferror(fp)) {
      printf(" Ошибка чтения выходного файла\n");
      break;
    }
    /* ожидание готовности порта-приемника */
    if(!feof(fp)) {
      wait(PORT);
      sport(PORT, ch);
    }
  } while(!feof(fp));
  wait(PORT);/* чтение последней порции из порта*/
  fclose(fp);
}
/* Возвращение значения длины файла в байтах */
unsigned int filesize(fp)
FILE *fp;
{
  unsigned long int i;
  i = 0;
  do {
    getc(fp);
```

```
    i++;
  } while(!feof(fp));
  rewind(fp);
  return (i-1); /* Не считая символ EOF */
}
/* Перекачка имени файла */
void send_file_name(f)
char *f;
{
  do {
     sport(PORT, '?');
  } while(!kbhit() && !(check_stat(PORT)&256));
  if(kbhit()) {
    getch();
    exit(1);
  }
  wait(PORT);
  while(*f) {
    sport(PORT, *f++);
    wait(PORT);
  }
  sport(PORT, '\0'); /* символ конца строки */
  wait(PORT);
}
/* ожидание ответа */
void wait(port)
int port;
{
  if(rport(port)!='.') {
    printf("Ошибка установления связи \n");
    exit(1);
  }
}
/* Передача символа из последовательного порта */
void sport(port, c)
int port; /* порт ввода/вывода */
char c;   /* пересылаемый символ */
{
  union REGS r;
  r.x.dx = port; /* последовательный порт */
  r.h.al = c; /* символ для передачи */
  r.h.ah = 1; /* функция передачи символа */
  int86(0x14, &r, &r);
  if(r.h.ah & 128) {
  printf("Ошибка передачи в последовательном порту %d",r.h.ah);
    exit(1);
  }
}


/* чтение символа из последовательного порта */
rport(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  /* ожидание символа */
  while(!(check_stat(PORT)&256))
    if(kbhit()) {
      getch();
      exit(1);
    }
  r.x.dx = port; /* последовательный порт */
  r.h.ah = 2; /* функция чтения символа  */
```

```
  int86(0x14, &r, &r);
  if(r.h.ah & 128)
    printf(" ошибка чтения в последовательном порту ");
  return r.h.al;
}
/* контроль состояния последовательного порта */
cheek_stat(port)
int port; /* порт ввода/вывода */
{
  union REGS r;
  r.x.dx = port; /* последовательный порт  */
  r.h.ah = 3;    /* чтение состояния */
  int86(0x14, &r, &r);
  return r.x.ax;
}

/* инициализация порта параметрами:
          скорость передачи - 9600 бод,
          контроль четности       выкл. ,
          восемь бит данных,
          два завершающих стоп-бита.
*/
void port_init(port)
int port;
{
  union REGS r;
  r.x.dx = port; /* последовательный порт */
  r.h.ah = 0;    /* функция инициализации порта*/
  r.h.al = 231; /* код инициализации - см. выше */
  int86(0x14, &r, &r);
}
```

ИСПОЛЬЗОВАНИЕ ЛВС

Для обеспечения функционирования ЛВС необходимо запустить файловый сервер на центральном компьютере. Каждая рабочая станция - абонент сети должна иметь в составе своего програмного обеспечения файлы GET.EXE и PUT.EXE. При необходимости получить файл, вводится команда GET, при необходимости сохранить файл во внешней памяти файлового сервера вводится команда PUT.

СОВЕРШЕНСТВОВАНИЕ ЛВС

Одним из первых усовершенствований описанной здесь простейшей ЛВС является обеспечение дополнительной возможности для рабочих станций сети оперировать с каталогом файловой системы центрального компьютера. Для этой цели может быть добавлена команда 'd' (directory) в набор командных примитивов сети. В простейшем случае обработка каталога сводится к его выдаче в виде перечня файлов текущей директории. Поэтому, исходя из вышеприведенного положения, добавление команды 'd' потребует соответствующего дополнения основного цикла работы файлового сервера с целью обеспечения выдачи каталога при передаче абонентом этой команды. Результат выполнения команды отображается обычным способом на экране так, будто вы выполнили команду dir на своем компьютере.

Довольно привлекательно выглядит расширение набора командных примитивов сети за счет внесения в него команды RUN, позволяющей автономно пересылать из файлового сервера выполняемый файл, размещать его в памяти рабочей станции и запускать.

Электронная почта, с помощью которой пользователи могут обмениваться друг с другом сообщениями, является одним из

перспективных направлений совершенствования сети.

В конечном итоге вы можете обеспечить защиту всей вашей системы путем разрешения загрузки для каждого узла сети (рабочей станции) лишь определенных файлов для защиты всей совокупности.

ИНТЕРПРЕТАТОРЫ ЯЗЫКА.

Вы когда-нибудь хотели создать свой язык программирования? Большинство программистов призывают к поиску идеи создания, управления и модификации своих языков программирования. Однако, лишь немногие программисты могут легко и непринужденно создать язык программирования. Создание полного компилятора является многообязывающей задачей, но гораздо проще создать интерпретатор языка. Методы, используемые для создания итерпретаторов языка, изучаются редко или изучаются довольно абстрактно. В этой главе на практических примерах вы будете изучать секреты интерпретации языка и грамматического разбора выражений.

Интерпретаторы важны по трем очень важным причинам. Во-первых, они могут обеспечивать удобную интерактивную среду (как доказательство - интерпретатор стандартного языка BASIC, которыми снабжаются большинство персональных компьтеров). Многие пользователи-новички находят, что интерактивная среда более удобна, чем компилятор. Во-вторых, интерпретаторы языка обеспечивают превосходные интерактивные отладочные возможности. Даже ветераны-программисты при отладке трудных программ прибегают к помощи интерпретатора языка, потому что он позволяет динамично устанавливать значения переменных и условий. В-третьих, большинство языков запросов к базе данных работают в режиме интерпретации.

В этой главе будет разработан интерпретатор для подмножества языка BASIC, который еще называют "SMALL BASIC". BASIC выбран вместо Си, потому что BASIC легче интерпретируется, чем Си или другой структурный язык. Интерпретатор для структурного языка, такого как Си более труден, чем для BASIC из-за автономных функций с локальными переменными, которые обеспечивают интерпретатору многозначность. Принципы, используемые в интерпретаторе BASIC, применимы и для других языков, и вы можете использовать написанные программы в качестве отправной точки. Прежде, чем начать работу, необходимо изучить сущность языковой интерпретации, особенно перед тем, как браться за интерпетацию такого сложного языка, как Си. Если вы не знаете BASIC, не беспокойтесь, команды используемые в SMALL BASIC очень легкие для понимания.

Мы начинаем с сердца любого интерпретатора: синтаксического анализатора выражений.

СИНТАКСИЧЕСКИЙ РАЗБОР ВЫРАЖЕНИЙ

Наиболее важной частью интерпретатора языка является синтактический анализатор выражений, который преобразует числовые выражения, такие как (10-X)/23, в такую форму, чтобы компьютер мог понять ее и вычислить. В книге по языку Си: The Complete Reference (Osborne/McGraw-uill, 1987) вступительная глава посвящена синтаксическому анализу выражений. Подобного же рода синтаксический анализ, основанный на принципах, изложенных в вышеупомянутой книге, (правда, с небольшими изменениями) будет использоваться для построения интерпретатора SMALL BASIC в данной главе нашей книги. (Так как эта глава содержит только краткие сведения о синтаксическом анализе выражений, то для более детального изучения этой проблемы советуем вам обратиться к источнику: The Compelete Reference.

Синтаксический анализ выражений является довольно сложной

задачей, однако в некоторых случаях она облегчается тем, что в процессе синтаксического анализа используются довольно строгие правила алгебры. Синтаксический анализатор, описанный в этой главе, в общем может быть классифицирован как синтаксический анализатор рекурсивного спуска.

Перед тем, как приступить к детальной разработке синтаксического анализатора, вы должны иметь представление о выражениях. Поэтому наш следующий параграф посвящен именно этому вопросу.

ВЫРАЖЕНИЯ

Хотя выражения могут быть составлены в принципе из любых типов данных, в этой главе мы будем иметь дело только с числовыми выражениями. Будем считать, что для наших целей числовые выражения могут строится из следующих элементов:

- числа
- операторы + - / * ^ % = () <> ; ,
- скобки
- переменные

Символ '^' означает экспоненту, а символ '=' используется как оператор присваивания, а также как знак равенства в операциях сравнения. Элементы выражения можно комбинировать согласно правилам алгебры.

Вот некоторые примеры выражений:

```
7-8 (100-5)*14/6
a+b-c
10^5
a=7-b
```

Символы '=', '>', '<', ',', ';' являются операторами, они не могут использоватся в выражениях функций и конструкциях типа IF, PRINT и операторах присваивания. (Заметим, что анализатор языка Си должен обрабатывать и эти операторы в различных их комбинациях).

Что касается языка BASIC, старшинство операторов не определено. В процессе работы синтаксический анализатор присваивает операторам следующие приоритеты:

```
высший      ()
            ^
            *  /  %
            +  -
низший      =
```

Операторы равного приоритета выполняются слева направо.

Синтаксис языка SMALL BASIC предполагает, что все переменные обозначаются одной буквой. Это позволяет оперировать в программе двадцати шестью переменными (буквы от A до Z). Хотя интерпретаторы языка BASIC поддерживают обычно большее число символов в определении переменной, (например, переменные типа X27), но для простоты изложения основных принципов построения интерпретаторов наш интерпретатор языка SMALL BASIC этого делать не будет. Будем считать также, что переменные разных регистров не отличаются друг от друга и, поэтому, переменные "a" и "A" будут трактоваться как одна и та же переменная. Условимся, что все числа являются целыми, хотя вы без особого труда можете написать программы для манипулирования другими типами чисел. Хотя символьные переменные в нашей версии языка и не поддерживаются, будем считать, что возможен вывод ограниченных символьных констант на экран в виде различных сообщений.

Итак, будем строить синтаксический анализатор исходя из перечисленных выше допущений. Теперь давайте рассмотрим такое базовое понятие теории синтаксического анализа как лексема.

ЛЕКСЕМЫ

Перед тем, как построить синтаксический анализатор, разбирающий значения выражений, вы должны иметь несколько вариантов разбиения строки, содержащей выражение, на составляющие части. Например, выражение

A*B-(W+10)

содержит компоненты "A", "*", "B", "-", "(", "W", "+", "10" и ")". Каждый компонент представляет собой неделимый элемент выражения. Такой компонент или независимая часть выражения называется лексемой. Функция, разбивающая выражение на составные части, должна решать четыре задачи: (1) игнорировать пробелы и символы табуляции, (2) извлекать каждую лексему из текста, (3) если необходимо, преобразовывать лексему во внутренний формат, (4) определять тип лексемы.

Каждая лексема имеет два формата: внешний и внутренний. Внешний формат - это символьная строка, с помощью которой вы пишите программы на каком-либо языке программирования. Например, "PRINT" - это внешняя форма команды PRINT языка BASIC. Можно построить интерпретатор из расчета, что каждая лексема используется во внешнем формате, но это типичное решение проблемы программистом-непрофессионалом, который лишь два часа назад оторвался от материной юбки и час назад увидел настоящий компьютер. Настоящие мужчины ориентируются на внутренний формат лексемы, который является просто числом, и разрабатывают интерпретаторы исходя из этой профессиональной точки зрения на проблему. Поясним этот подход. Например, команда PRINT может иметь порядковый внутренний номер 1, команда INPUT - 2 и т.д. Преимущество внутреннего формата заключается в том, что программы, обрабатывающие числа, более быстродействующие, чем программы, обрабатывающие строки. Для реализации такого подхода необходима функция, которая берет из входного потока данных очередную лексему и преобразует ее из внешнего формата во внутренний. Помните, что не все лексемы имеют разные форматы. Например, операторы не подлежат преобразованию потому, что они могут трактоваться как символы или числа в своих внешних форматах.

Очень важно знать, какой тип лексемы возвращен. Например, анализатору выражений необходимо знать, является ли следующая лексема числом, оператором или переменной. Значение типа лексемы для процесса анализа в целом станет очевидным, когда вы приступите непосредственно к разработке интерпретатора.

Функция, которая возвращает следующую лексему в выражении, называется get_token( ). Она работает из расчета того, что в языке SMALL BASIC, программа хранится как одна строка, ограниченная в конце символом завершения строки (\0). Функция get_token() сканирует текст программы, анализируя по одному символу, при этом глобальный указатель анализатора принимает

значение адреса очередной считаной лексемы. В версии get_token(), приведенной ниже, этот указатель называется prog. Так как prog является глобальной переменной, то его значение между вызовами get_token сохраняется и позволяет другим функциям использовать его.

Анализатор, разрабатываемый в этой главе, использует шесть типов лексем: DELIMITER, VARIABLE, NUMBER, COMMAND, STRING и QUOTE (разделитель, переменная, число, команда, строка и кавычки). Тип VARIABLE приписывается переменным. Тип DELIMITER приписывается операторам и скобкам. Тип NUMBER - для чисел. Тип COMMAND - для команд языка SMALL BASIC. Тип STRING временно используется внутри get_token() пока идет разбор лексемы. Тип QUOTE используется при определении кавычек, ограничивающих строку. Глобальная переменная token_type содержит тип лексемы. Внутреннее представление лексемы помещается в глобальную переменную tok.

Ниже приведена функция get_token(). Все остальные необходимые вспомогательные функции для полного синтаксического анализатора будут приведены в этой главе немного позже.

```
#define DELIMITER  1
#define VARIABLE   2
#define NUMBER     3
#define COMMAND    4
#define STRING     5
#define QUOTE      6
#define FINISHED   10
#define EOL        9
extern char token[80];
extern int tok, token_type;
extern char *prog;  /* Содержит анализируемое выражение */
/* Получить лексему */
get_token()
{
  register char *temp;
  token_type=0; tok=0;
  temp=token;
  if(*prog=='\0')  { /* Конец файла */
    *token=0;
    tok=FINISHED;
    return(token_type=DELIMITER);
   }
   while(iswhite(*prog)) ++prog;  /* пропуск пробелов */
   if(*prog=='\r') { /* crtl */
     ++prog; ++prog;
     tok= EOL; *token='\r';

     token[1]='\n';token[2]=0;
     return (token_type = DELIMITER);
   }
   if(strchr("+-*^/%=;(),><", *prog)) { /* разделитель */
     *temp=*prog;
     prog++; /* переход на слкдующую позицию */
     temp++;
     *temp=0;
     return (token_type=DELIMITER);
   }
   if(*prog=='"')  { /* строка в кавычках */
    prog++;
     while(*prog != '"' && *prog!='\r') *temp++=*prog++;
     if(*prog=='\r') serror(1);
     prog++;*temp=0;
     return(token_type=QUOTE);
   }
   if(isdigit(*prog)) { /* число */
     while(!isdelim(*prog)) *temp++=*prog++;
     *temp = '\0';
     return(token_type = NUMBER);
   }
   if(isalpha(*prog))  { /* переменная или команда */
     while(!isdelim(*prog)) *temp++=*prog++;
     token_type=STRING;
   }
   *temp = '\0';
/* Просматривается, если строка есть команда или переменная */
   if(token_type==STRING) {
     tok=look_up(token); /* преобразование во внутренний
                            формат */
     if(!tok) token_type = VARIABLE;
     else token_type = COMMAND; /* это команда */
   }
```

```
    return token_type;
  }
```

Посмотрите внимательно на get_token(). Многие программисты любят помещать пробелы перед выражениями для улучшения удобочитаемости и наглядности своей программы. Лидирующие пробелы пропускаются с помощью функции is_white(), которая возвращает значение "истина" ("TRUE"), если ее аргумент является пробелом или символом табуляции. Псле пропуска пробелов, сканер, реализуемый с помощью программы prog, указывает на каждое число, переменную, команду, символ "возврат каретки" или ноль, если достигнут конец выражения (программы). Если очередным анализируемым символом является символ "возврат каретки" (\r), то возвращается значение "конец строки программы" ("EOL"). Если

очередной символ является оператором, то в качестве значения глобальной переменной token возвращается соответствующая строка, при этом в переменную token_type помещается значение DELIMITER. В противном случае проверяется наличие кавычек. Затем происходит проверка является ли лексема числом. Если лексема является символом, то она, следовательно, является или переменной или командой. Функция look_up() сравнивает внешний формат лексемы с таблицей лексем, определенной при разработке анализатора и, если находит соответствующе значение в ней, возвращает внутреннее представление лексемы (команды). В противном случае лексема трактуется как переменная. И, наконец, если символ не удовлетворяет ни одному из условий, приведенных выше, то он трактуется как символ конца выражения. При этом значение token обнуляется.

Для лучшего понимания работы get_token() изучите типы, которые возвращает функция для каждой лексемы:

PRINT A+100-(B*C)/2

Лексема	Тип лексемы.
PRINT	COMMAND
A	VARIABLE
+	DELIMITER
100	NUMBER
-	DELIMITER
(	DELIMITER
B	VARIABLE
*	DELIMITER
C	VARIABLE
)	DELIMITER
/	DELIMITER
2	NUMBER
null	DELIMITER

Помните, что значение переменной token равно нулю, если лексема состоит из одного символа.

Некоторые функции интерпретатора нуждаются в повторном просмотре лексемы. В этом случае лексема должна быть возвращена во входной поток. Функция putback() решает эту задачу.

```
/*  Возвращает лексему обратно во входной поток */
    void putback()
    {
      char *t;
      t = token;
      for(; *t; t++) prog--;
    }
```

ПОРЯДОК ПОСТРОЕНИЯ ВЫРАЖЕНИЙ

Имеется много вариантов анализа и вычисления выражений. Для

использования полного синтаксического анализатора рекурсивного спуска мы должны представить выражение в виде рекурсивной структуры данных. Это означает, что выражение определяется в термах самого себя. Если выражение можно определить с использованием только символов "+" ,"-" ,"*" ,"/" и скобок, то все выражения могут быть определены с использованием следующих правил:

Выражение = > Терм [+Терм][-Терм]
Терм = > Фактор [*Фактор][/Фактор]
Фактор = > Переменная, Число или (Выражение)

Очевидно, что некоторые части в выражении могут отсутствовать вообще. Квадратные скобки означают именно такие необязательные элементы выражения. Символ => имеет смысл "продуцирует".

Фактически, выше перечислены правила, которые обычно называют правилами вывода выражения. В соответствии с этими правилами терм можно определить так: "Терм является произведением или отношением факторов".

Вы вероятно заметили, что приоритет операторов безусловен в описанных выражениях, то есть вложенные элементы включают операторы с более высоким приоритетом.

В связи с этим рассмотрим ряд примеров. Выражение

10+5*B

содержит два терма: "10" и "5*B". Они, в свою очередь, состоят из трех факторов: "10", "5" и "B", содержащих два числа и одну переменную.

В другом случае выражение

14*(7-C)

содержит два фактора "14" и "(7-C)", которые, в свою очередь, состоят из числа и выражения в скобках. Выражение в скобках вычисляется как разность числа и переменной.

Можно преобразовать правила вывода выражений в множество общих рекурсивных функций, что и является зачастую основной формой синтаксического анализатора рекурсивного спуска. На каждом шаге анализатор такого типа выполняет специфические операции в соответствии с установленными алгебраическими правилами. Работу этого процесса можно рассмотреть на примере анализа выражения и выполнения арифметических операций.

Пусть на вход анализатора поступает следующее выражение:

9/3-(100+56)

Анализатор в этом случае будет работать по такой схеме:

1. Берем первый терм: "9/3".
2. Берем каждый фактор и выполняем деление чисел, получаем результат "3".
3. Берем второй терм: "(100+56)". В этой точке стартует рекурсивный анализ второго выражения.
4. Берем каждый фактор и суммируем их между собой, получаем результат 156
5. Берем число, вернувшееся из рекурсии, и вычитаем его из первого: 3-156. Получаем итоговый результат "-153".

Если вы немного смущены столь сложной схемой работы анализатора, то уверяем вас, что это не так уж страшно. Гораздо страшнее оказаться у телевизора, когда транслируют финальный футбольный матч, не имея с собой достаточного запаса пива. Поэтому не пугайтесь комплексного подхода.

Вы должны помнить две основные идеи рекурсивного разбора выражений: (1) приоритет операторов является безусловным в продукционных правилах и определен в них; (2) этот метод синтаксического анализа и вычисления выражений очень похож на тот, который вы сами используете для выполнения таких же операций.

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР ВЫРАЖЕНИЙ

Полный простой синтаксический анализатор рекурсивного спуска для целых числовых выражений включает в себя ряд функций. Вы должны взять тексты этих функций и сохранить их в своем файле (когда тексты анализатора и интерпретатора объединятся получится довольно большой файл, поэтому рекомендуется откомпилировать файлы отдельно). Смысл использования глобальных переменных будет кратко описан, в процессе обсуждения интерпретатора.

Исходный текст простейшего синтаксического анализатора рекурсивного спуска для целочисленных выражений приведен ниже.

```
/* Синтаксический анализатор рекурсивного спуска
   для целочисленных выражений, который содержит
   ряд включаемых переменных
*/
   #include "setjmp.h"
   #include "math.h"
   #include "ctype.h"
   #include "stdlib.h"
   #define  DELIMITER 1
   #define  VARIABLE  2
   #define  NUMBER    3
   #define  COMMAND   4
   #define  STRING    5
   #define  QUOTE     6
   #define  EOL       9
   #define  FINISHED  10
   extern char *prog;  /* буфер анализируемого выражения */
   extern jmp_buf e_buf; /* буфер среды функции longjmp() */
   extern int variables[26]; /* переменные */
     extern struct commands {
     char command[20];
     char tok;
   } table[];
  extern char token[80]; /* внешнее представление лексемы */
  extern char token_type; /* тип лексемы */
  extern char tok; /* внутреннее представление лексемы */
    void get_exp(),level2(),level3(),level4(),level5();
    void level6(),primitive(),arith(),unary();
    void serror(), putback();
    /* Точка входа в анализатор. */
    void get_exp(result)
    int *result;
    {
      get_token();
      if(!*token) {
        serror(2);

        return;
      }
      level2(result);
      putback(); /* возвращает последнюю считаную
                    лексему обратно во входной поток */
    }
    /* Сложение или вычитание двух термов */
    void level2(result)
    int *result;
    {
      register char op;
      int hold;
      level3(result);
      while((op=*token) == '+' || op == '-') {
        get_token();
        level3(&hold);
        arith(op,result,&hold);
```

```
    }
  }
 /* Вычисление произведения или частного двух фвкторов */
  void level3(result)
  int *result;
  {
    register char op;
    int hold;
    level4(result);
    while((op = *token) == '+' || op == '/' || op == '%') {
      get_token();
      level4(&hold);
      arith(op,result,&hold);
    }
  }
  /* Обработка степени числа (целочисленной) */
  void level4(result)
  int *result;
  {
    int hold;
    level5(result);
    if(*token== '^') {
      get_token();
      level4(&hold);
      arith('^', result, &hold);
    }
  }
  /* Унарный + или - */
  void level5(result)

  int *result;
  {
    register char op;
    op = 0;
if((token_type==DELIMITER) && *token=='+' || *token=='-') {
      op = *token;
      get_token();
    }
    level6(result);
    if(op)
      unary(op, result);
  }
  /* Обработка выражения в круглых скобках */
  void level6(result)
  int *result;
  {
    if((*token == '(') && (token_type == DELIMITER)) {
      get_token();
      level2(result);
      if(*token != ')')
        serror(1);
      get_token();
    }
    else
      primitive(result);
  }
  /* Определение значения переменной по ее имени */
  void primitive(result)
  int *result;
  {
    switch(token_type) {
    case VARIABLE:
      *result = find_var(token);
      get_token();
```

```
    return;
  case NUMBER:
    *result  = atoi(token);
    get_token();
    return;
  default:
    serror(0);
  }
}
/* Выполнение специфицированной арифметики */
void arith(o, r, h)
char o;
int *r, *h;
{register int t, ex;
switch(o) {

  case '-':
    *r = *r-*h;
    break;
  case '+':
    *r = *r+*h;
    break;
  case '*':
    *r = *r * *h;
    break;
  case '/':
    *r = (*r)/(*h);
    break;
  case '%':
    t = (*r)/(*h);
    *r = *r-(t*(*h));
    break;
  case '^':
    ex =*r;
    if(*h==0) {
      *r = 1;
      break;
    }
    for(t=*h-1; t>0; --t) *r = (*r) * ex;
    break;
  }
}
/* Изменение знака */
void unary(o, r)
char o;
int *r;
{
  if(o=='-') *r = -(*r);
}
/* Поиск значения переменной */
int find_var(s)
char *s;
{
  if(!isalpha(*s)){
    serror(4); /* не переменная */
    return 0;
  }
  return variables[toupper(*token)-'^'];
}
/* выдать сообщение об ошибке */
void serror(error)
int error;
{
  static char *e[]= {
```

```
        "Синтаксическая ошибка",
        "Непарные круглые скобки",
        "Это не выражениет",

        "Предполагается символ равенства",
        "Не переменная",
        "Таблица меток переполнена",
        "Дублирование меток",
        "Неопределенная метка",
        "Необходим оператор THEN",
        "Необходим оператор TO",
        "Уровень вложенности цикла FOR слишком велик",
        "NEXT не соответствует FOR",
        "Уровень вложенности GOSUB слишком велик",
        "RETURN не соответствует GOSUB"
      };
      printf("&4%s\n",e[error]);
      longjmp(e_buf, 1); /* возврат в точку сохранения */
    }
    /* Чтение лексемы. */
    get_token()
    {
      register char *temp;
      token_type=0; tok=0;
      temp=token;
      if(*prog=='\0') { /* Конец файла */
        *token=0;
        tok = FINISHED;
        return(token_type=DELIMITER);
      }
      while(iswhite(*prog)) ++prog; /* пропуск пробелов */
      if(*prog=='\r') { /* коней строки программы */
        ++prog; ++prog;
        tok = EOL; *token='\r';
        token[1]='\n'; token[2]=0;
        return (token_type = DELIMITER);
      }
      if(strchr("+-^/%=;(),><", *prog)){ /* разделитель */
        *temp=*prog;
        prog++; /* переход на следующую позицию */
        temp++;
        *temp=0;
        return (token_type=DELIMITER);
      }
      if(*prog=='"') { /* строка кавычек */
        prog++;
        while(*prog != '"' && *prog!='\r') *temp++=*prog++;
        if(*prog=='\r') serror(1);
        prog++;*temp=0;
        return(token_type=QUOTE);
      }
      if(isdigit(*prog)) { /* число */

        while(!isdelim(*prog)) *temp++=*prog++;
        *temp = '\0';
        return(token_type = NUMBER);
      }
      if(isalpha(*prog)) { /* переменная или команда */
        while(!isdelim(*prog)) *temp++=*prog++;
        token_type=STRING;
      }
      *temp = '\0';
/* просматривается, если строка - переменная или команда */
      if(token_type==STRING) {
```

```
  tok=look_up(token); /* Преобразование во внутренний формат */
            if(!tok) token_type = VARIABLE;
            else token_type = COMMAND; /* это команда */
          }
          return token_type;
        }
        /* Возврат лексемы во входной поток */
       void putback()
        {
          char *t;
          t = token;
          for(; *t; t++) prog--;
        }
        /* Поиск соответствия внутреннего формата для
           текущей лексемы в таблице лексем.
        */
        look_up(s)
        char *s;
        {
          register int i,j;
          char *p;
          /* преобразование к нижнему регистру */
          p = s;
          while(*p){ *p = tolower(*p); p++; }
          /* просматривается, если лексема обнаружена в
             таблице */
          for(i=0; *table[i].command; i++)
            if(!strcmp(table[i].command, s)) return table[i].tok;
          return 0; /* нераспознанная команда */
        }
        /* Возвращает "истину", если "c" разделитель */
        isdelim(c)
        char c;

        {
        if(strchr(" ;,+-<>/*%^=()",c) || c==9 || c=='\r' || c==0)
            return 1;
          return 0;
        }
        /* Возвращает 1, если "c" пробел или табуляция */
        iswhite(c)
        char c;
        {
          if(c==' ' || c=='\t') return 1;
          else return 0;
        }
```

Анализатор поддерживает следующие операторы: "+", "-", "*", "/", "%", целочисленный показатель степени (^) и унарный минус. Все эти операции могут использоваться в сочетании с круглыми скобками. Вы, вероятно, заметили, что в программе имеются функции шести уровней, которые работают точно также как функция primitive(), вращающая значение числа. Помимо этих функций, реализующих арифметические операции, в состав программы включены функции arith() и unary(), а также get_token().

При вычислении выражения prog указывает на начало строки, содержащей выражение, и вызывает get_exp() с адресом переменной, в которую вы хотите поместить результат.

Обратите особое внимание на функцию serror(), которая используется для выдачи сообщений об ошибках. При обнаружении синтаксической ошибки serror() вызывается с номером этой ошибки в качестве аргумента. Ошибка с кодом 0, которой соответствует сообщение "синтаксическая ошибка", выдается в том случае, когда тип ошибки нельзя определить более конкретно. В других случаях ошибка уточняется. Заметьте, что serror() заканчивается вызовом

функции longjmp().

Функция logjmp() выполняет нелокальный переход, возвращаясь в точку, определенную с помощью функции setjmp(). Функция setjmp() включена в исходный текст интерпретатора. Первый аргумент функции logjmp() является буфером среды, инициированной с помощью setjmp(). Второй аргумент - это значение, которое определяется при передаче управления из setjmp() обратно в точку ее вызова. Как это делается вы, увидите позже.

Использование ljgjmp() упрощает обработку ошибок, так как программы-анализаторы не должны аварийно завершаться при обнаружении ошибки. Если ваш компилятор Си не поддерживает функции setjmp() и logjmp(), то каждая функция при обнаружении ошибок должна выполнять возврат в точку возникновения ошибки самостоятельно.

КАК АНАЛИЗАТОР ОБРАБАТЫВАЕТ ПЕРЕМЕННЫЕ

Как было сказано раньше, интерпретатор языка SMALL BASIC распознает переменные с именами только от "A" до "Z". Каждой переменной соответствует элемент массива variables, состоящего из 26 элементов. Этот массив определен в тексте интерпретатора, как показано ниже, и инициализируется нулевыми значениями.

```
int variables[26]= {   /* 26 переменных пользователя, A-Z */
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0
};
```

Так как именами переменных являются буквы от "A" до "Z", то индексирование массива variables можно легко осуществить путем вычитания из соответствующих значений имен переменных в коде ASCII кода символа 'A'. Функция find_var(), определяющая значение переменной в зависимости от ее имени, представлена ниже.

```
/* Определение значения переменной по ее имени*/
int find_var(s)
char *s;
{
  if(!isalpha(*s)){
    serror(4); /* это не переменная */
    return 0;
  }
  return variables[toupper(*token)-'A'];
}
```

Эта функция допускает использование более длинных имен, но только первая буква имени переменной является значащей.

ИНТЕРПРЕТАТОР ЯЗЫКА SMALL BASIC

Разрабатываемый интерпретатор будет распознавать следующие ключевые слова языка программирования BASIC:

```
PRINT
INPUT
IF
THEN
FOR
NEXT
TO
GOTO
GOSUB
RETURN
END
```

Внутреннее представление этих команд (плюс значение EOL для конца строки и FINISHED для сигнализации о конце программы) определяется так:

```
#define PRINT   1
```

```
#define INPUT   2
#define IF      3
#define THEN    4
#define FOR     5
#define NEXT    6
#define TO      7
#define GOTO    8
#define EOL     9
#define FINISHED 10
#define GOSUB    11
#define RETURN   12
#define END      13
```

Для преобразования внешнего представления лексем во внутренний формат используется вспомагательная структура table.

```
struct commands { /* Вспомогательная структура ключевых
                     слов анализатора                  */
  char command[20];
  char tok;
} table[] = { /* Таблица обрабатывает команды, введенные */
  "print",PRINT, /* на нижнем регистре */
  "input",INPUT,
  "if",IF,
  "then",THEN,
  "goto",GOTO,
  "for",FOR,
  "next",NEXT,
  "to",TO,
  "gosub",GOSUB,
  "return",RETURN,

  "end",END,
  "",END  /* mark end of table */
};
```

Обратите внимание на то, что признак конца файла (нулевая строка) помещен в конец таблицы.

Функция look_up() возвращает внутреннее представление каждой лексемы или символа '\0', если таковая не обнаружена.

```
/* Преобразование каждой лексемы из таблицы лексем
   во внутреннее представление.
*/
look_up(s)
char *s;
{
  register int i,j;
  char *p;
  /* преобразование в символы нижнего регистра */
  p =s;
  while(*p){ *p = tolower(*p); p++; }
  /* если лексема обнаружена в таблице */
  for(i=0; *table[i].command; i++)
      if(!strcmp(table[i].command, s)) return table[i].tok;
  return 0; /* команда не распознана */
}
```

Интерпретатор языка SMALL BASIC не поддерживает редактор текстов, поэтому вы должны создавать программы на языке BASIC, используя стандартный текстовый редактор.

Каждая программа считывается и выполняется с помощью интерпретатора. Функция, которая загружает программу, называется load_program().

```
/* Загрузка программы */
load_program(p, fname)
char *p;
char *fname;
{
```

```
FILE *fp;
int i=0;
if(!(fp=fopen(fname, "rb"))) return 0;
i = 0;
  do {
     *p = getc(fp);
     p++; i++;
  } while(!feof(fp) && i<PROG_SIZE);
  *(p-2) = '\0'; /* Символ конца загружаемой программы */
  fclose(fp);
  return 1;

}
```

ОСНОВНОЙ ЦИКЛ РАБОТЫ АНАЛИЗАТОРА

Все интерпретаторы выполняют операции путем считывания лексемы программы и выбора необходимой функции для ее выполнения. Основной цикл работы для интерпретатора языка SMALL BASIC выглядит следующим образом.

```
do {
  token_type = get_token();
  /* Проверка на соответствие оператору языка */
  if(token_type == VARIABLE) {
    putback(); /* возврат переменной во входной поток */
    assignment(); /* должен быть оператор присваивания */
  }
  else /* это команда */
    switch(tok) {
      case PRINT:
        print();
        break;
      case GOTO:
        exec_if();
        break;
      case FOR:
        exec_for();
        break;
      case NEXT:
        next();
        break;
      case INPUT:
        input();
        break;
      case GOSUB:
        gosub();
        break;
      case RETURN:
        greturn();
        break;
      case END:
        exit(0);
    }
} while (tok != FINISHED);
```

Сначала лексема считывается из программы. Для удобства анализа каждая лексема располагается на отдельной строке. Если лексема является переменной, то, следуя синтаксису языка, за ней должен следовать оператор присваивания (SMALL BASIC не поддерживает старомодную команду LET). В противном случае, лексема считается командой и с помощью оператора **case** в зависимости от значения tok происходит выбор соответствующей

команды. Посмотрите, как работает каждая из них.

КОМАНДА ПРИСВАИВАНИЯ ЗНАЧЕНИЙ.

В языке BASIC основной формой оператора присваивания является следующая:

<имя переменной>=<выражение>

Функция assignment() поддерживает этот тип присваивания.

```
 /* Присвоить значение переменной */
 assignment()
 {
   int var, value;
   /* получить имя переменной */
   get_token();
     if(!isalpha(*token)) {
       serror(4); /* это не переменная */
       return;
     }
     /* поиск индекса переменной в массиве */
     var = toupper(*token)-'A';
     /* считать символ равенства*/
     get_token();
     if(*token!='=') {}
       serror(3);
       return;
     }
     /* считать присваемое переменной значение */
     get_exp(&value);
     /* присвоить значение*/
     variables[var] = value;
   }
```

Сначала assignment() считывает лексему из программы. Это - лексема-переменная, которой должна быть присвоена величина. Если лексема не является переменной, то сообщается об ошибке. Затем считывается знак равенства (очередная лексема). Далее вызывается функция get_exp(), которая вычисляет значение переменной. Функция получается удивительно простой потому, что анализатор выражений и get_token делают большую часть рутинной работы.

КОМАНДА PRINT

Команда PRINT стандарта языка BASIC достаточно мощная и гибкая, особенно когда применяется ее формат PRINT USING. Хотя создание функции, которая реализует все возможности этой команды, выходит за рамки этой главы, разрабатываемая функция является наиболее важной. Вот основная форма команды PRINT языка SMALL BASIC:

PRINT <список аргумеентов>

где <список аргументов> представляет собой перечень переменных, заключенных в кавычки и разделенных запятыми или точкой с запятой. Функция print(), представленная ниже, является аналогом команды PRINT языка BASIC. Обратите внимание, что печать выполняется в теле цикла **do-while**, условием завершения которого является вывод на печать всего списка аргументов команды.

```
/* Простейшая версия оператора PRINT */
void print()
{
  int answer;
  int len=0, spaces;
  char last_delim;
  do {
       get_token(); /* получить следующий элемент списка */
       if(tok == EOL || tok == FINISHED) break;
       if(token_tipe == QUOTE) { /* это строка */
           printf(token);
             len += strlen(token);
```

```
            get_token();
        }
         else { /* это выражение */
            putback();
            get_exp(&answer);
            get_token();
            len+= printf("%d", answer);
         }
         last_delim = *token;
         if(*token == ';') {
    /* Вычисление числа пробелов при переходе к следующей
       табуляции */
         spaces = 8-(len % 8);
         len += spaces; /* смещение на одну табуляцию */
         while (spaces) {
           printf(" ");
           spaces--;
         }
         }
         else if(*token == ','); /* ничего не делать */;
         else if(tok != EOL && tok != FINESHED) serror(0);
         } while (*token == ';' || *token == ',');

         if(tok == EOL || tok == FINESHED) {
            if(last_delim != ';' && last_delim != ',')
                                                 printf("\n");
            else serror(0); /* Отсутствует разделитель */
        }
```

Команда PRINT может быть использована для вывода списка переменных и строчных констант на экран. Если два элемента разделены запятой, то их значения выводятся на печать без пробелов (конкатенируются). Если же два элемента разделены точкой с запятой, то второй элемент выводится начиная, со следующей позиции табуляции. Если список элементов заканчивается запятой или точкой с запятой, то переход на новую строку не выполняется.

Приведенные ниже примеры вызовут печать данных с новой строки:

```
PRINT X; Y; "ЭТО СТРОКА"
PRINT 10 / 4
PRINT
```

Функция print() использует функцию putback() для возврата лексемы во входной поток. Причиной этого является то, что прежде, чем начать печать строки, заключенной в скобки, или числового выражения, функция print() должна проанализировать следующий элемент списка аргументов. Если следующий элемент является выражением, то первый терм выражения должен быть помещен обратно во входной поток, так как в противном случае анализатору не удастся корректно обработать это выражение.

КОМАНДА INPUT.

В языке BASIC команда INPUT используется для чтения информации с клавиатуры и сохранения ее в переменных. Она имеет два основных формата. Первый формат команды выводит маркер ожидания ввода данных ('?') и переводит всю программу в ожидание ввода данных:

INPUT <имя переменной>

Второй формат приводит к отображению на экране строки символов, после чего ожидается ввод данных:

INPUT "<символьная строка>" <имя переменной>

Функция input (), приведенная ниже, реализует команду языка BASIC INPUT.

```
    /* Простейшая версия оператора INPUT */
    void input()
```

```
    {
      char str[80], var;
      int i;
      get_token(); /*Анализ наличия символьной строки */
      if(token_type == QUOTE) {
        printf(token); /*Если строка есть, проверка запятой */
        get_token();
        if(*token != ',') serror(1);
        get_token();
      }
      else printf("? "); /* В противном случае отображение "? "*/
 var = toupper(*token)-'A'; /* Вычисление индекса массива имен */
      scanf("%d",&i);   /* Чтение входных данных */
      variables[var] = i;  /* Сохранение их */
    }
```

Работа функции станет ясной и понятной после чтения комментариев.

КОМАНДА **GOTO**

Сейчас вы увидите реализацию одной из самых простых команд, но в то же время довольно сложной для разработчика. В языке BASIC основной формой программного управления является команда **GOTO**. В стандарте языка BASIC объектом, на который должен указывать **GOTO**, является номер строки. Этот традиционный подход сохраняется и в языке SMALL BASIC. Однако, SMALL BASIC не требуется нумеровать каждую строку. Номер строки необходим только в том случае, если на нее ссылается команда **GOTO**. Основной формат команды **GOTO** представлен ниже:

GOTO <номер строки>

Основной сложностью реализации оператора **GOTO** является то, что он должен позволять совершать переходы как вниз, так и вверх по программе. Для удовлетворения этого требования необходим механизм, который бы просматривал программу, выбирал каждую метку и помещал ее в таблицу, содержащую как имя метки так, и указатель ее размещения в программе. Такая таблица определена ниже.

```
    struct label {
      char name[LAB_LEN]; /* имя метки */
      char *p; /* указатель на место размещения в программе */
    };
    struct label label_table[NUM_LAB];
```

Функция, которая просматривает программу и выбирает каждую метку для размещения ее в таблице, называется scan_labels(). Она и основные поддерживающие ее функции приведены ниже.

```
   /* Поиск всех меток */
   void scan_labels()
   {
     register int loc;
     char *temp;
     label_init();  /* обнуление всех меток */
     temp = prog;   /* запись указателя на начало программы */
     /* Если первая лексема файла является меткой */
     get_token();
     if(token_type==NUMBER) {
       strcpy(label_table[0].name,token);
       label_table[0].p=prog;
     }
     find_eol();
     do {
       get_token();
       if(token_type==NUMBER) {
         loc=get_next_label(token);
         if(loc == -1 || loc == -2) {
             (loc == -1) ? serror(5):serror(6);
```

```
        }
        strcpy(label_table[loc].name, token);
      label_table[loc].p = prog; /* текущий указатель программы*/
      }
      /* Если строка не помечена, переход к следующей */
      if(tok!=EOL) find_eol();
    } while(tok != FINISHED);
    prog = temp; /* восстановить начальное значение*/
}
/* Инициализация массива хранения меток.
   По соглашению имя нулевой метки указывает, что
   данная позиция массива не используется  */
void label_init()
{
  register int t;
  for(t=0;t<NUM_LAB;++t) lable_table[t].name[0]='\0';
}
/* Поиск начала следующей строки */
void find_eol()
{
  while (*prog!='\n'  && *prog!='\0') ++prog;
  if(*prog) prog++;
}
/* Возвращает индекс следующей свободной позиции в массиве
   меток. Значение -1 указывает на переполнение массива.
   Значение -2 указывает на дублирование имен меток      */
get_next_label(s)
char *s;
{
  register int t;
  for (t=0;t<NUM_LAB;++t) {
    if(label_table[t].name[0] == 0) return t;
    if(!strcmp(label_table[t].name,s)) return -2; /* дубль */
  }
  return -1;
}
```

Функция scan_labels() сообщает о двух типах ошибок. Первый - это дублирующие метки. В языке BASIC (как и в большинстве других языков) не может быть двух одинаковых меток. Второй тип ошибки - это переполнение таблицы меток. Размер таблицы определен с помощью макроопределения NUM_LAB. Вы можете изменить его в ту или иную сторону.

Таблица меток строится один раз, и выполнение перехода **GOTO** выполняется с помощью функции exec_**goto()**, приведенной ниже.

```
/* Реализация оператора GOTO */
void exec_goto()

{
  char *loc;
  get_token(); /* получить метку перехода */
  /* Поиск местоположения метки в программе*/
  loc = find_label(token);
  if(loc == '\0')
    serror(7);   /* Метка не найдена  */
  else prog=loc; /*Программа начала выполнятся не с начала*/
}
/* Поиск местоположения метки. Ноль возвращается, если метка
   не найдена; в противном случае возвращается указатель на
   место, где расположена метка                              */
char *find_label(s)
char *s;
{
```

```
    register int t;
    for (t=0;t<NUM_LAB;++t)
     if(!strcmp(label_tabel[t].name,s)) return label_tabel[t].p;
    return '\0';
    }
```

Вспомогательная функция find_label(), получая метку, ищет ее в таблице меток и возвращает указатель на нее. Если метка не обнаружена, возвращает, значение null. Если адрес не null, то он присваивается переменной prog, что вызывает выполнение той строки, которая была помечена меткой. (Помните о том, что указатель prog описывает путь выполнения считаной программы). Если указатель не обнаружен, то выдается сообщение о том, что метка неопределена.

ОПЕРАТОР IF

Интерпретатор SMALL BASIC обрабатывает оператор IF в соответствии со стандартом языка BASIC. В SMALL BASIC, отсутствует ELSE и поддерживается только три условия: "больше", "меньше" или "равно". (Это сделано для того, чтобы вы могли легко понять работу этого оператора). Основной формат оператора:

IF <выражение><оператор><выражение> THEN <оператор>

Оператор, стоящий после THEN, выполняется только в том случае, если значение сравнения является истинным. Функция exec_if(), приведенная ниже, обеспечивает выполнение этой формы оператора IF.

```
    /* Простейшая реализация оператора IF  */
    void exec_if()
    {
      int x , y, cond;
      char op;
      get_exp(&x); /* получить левое выражение */
      get_token(); /* получить оператор  */
      if(!strchr("=<>", *token)) {
      serror(0);      /* недопустимый оператор  */
      return;
    }
    op=*token;
    get_exp(&y);  /* получить правое выражение */
    /* Определение результата */
    cond=0;
    switch(op) {
        case '=':
      if(x==y) cond=1;
      break;
        case '<':
      if(x<y) cond=1;
      break;
        case '>':
      if(x>y) cond=1;
      break;
    }
    if(cond) {  /* если истина, то поиск нужного IF  */
      get_token();
      if(tok!=THEN) {
         serror(8);
         return;
      }  /* В противном случае программа продолжается со
            следующей строки  */
    }
    else find_eol(); /* поиск строки продолжения программы */
  }
```

Функция exec_if() выполняется следующим образом:

1. Вычисляется значение левого выражения.
2. Считывается оператор сравнения.
3. Вычисляется величина правого выраженния.
4. Выполняется операция сравнения.
5. Если условие является истиной, то выполняется поиск THEN; в противном случае, find_eol выполняет переход на начало следующей строки.

ЦИКЛ FOR

Проблема обработки интерпретатором оператора цикла **FOR**, входящего в состав оператора BASIC, решена в нашем случае довольно оригинально. Основной формат оператора цикла **FOR** следующий:

FOR<имя управляющей переменной>=<нач. значение>TO<кон. значение>
.
.
.
последовательность операторов
.
.
.
NEXT

Версия оператора **FOR**, поддерживаемая интерпретатором SMALL BASIC, реализует цикл с положительным проращением равным 1 на каждую итерацию цикла. Параметр STEP не поддерживается.

В языке BASIC, точно также как и в Си, допускается вложенность цикла **FOR**. Основной изюминкой, при реализации этого оператора, с точки зрения программиста-профессионала, является сохранение информации о каждом вложенном цикле со ссылкой на внешний цикл. Для реализации этой маленькой, но все-таки заковырки **(**трудности всегда радуют настоящих программистов-мужчин**)**, используется стековая структура, которая работает следующим образом: Начало цикла, информация о значении управляющей переменной цикла и ее конечном значении, а также место расположения цикла в теле программы заносятся в стек. Каждый раз, при достижении оператора NEXT, из стека извлекается информация о значении управляющей переменной, затем ее значение пересчитывается и сравнивается с конечным значением цикла. Если значение управляющей переменной цикла достигло своего конечного значения, выполнение цикла прекращается и выполняется оператор программы следующий за оператором NEXT. В противном случае, в стек заносится новая информация и выполнение цикла начинается с его начала. Таким же образом обеспечивается интерпретация и выполнение вложенных циклов. В стекоподобной структуре вложенных циклов каждый **FOR** должен быть закрыт соответствующим оператором NEXT.

Для реализации оператора цикла **FOR** стек должен иметь следующую структуру:

```
struct for_stack {
     int var; /* счетчик цикла */
     int target; /* конечное значение */
     char *loc;
} fstack[FOR_NEST]; /* стек для цикла FOR/NEXT */
int ftos; /* индекс начала стека FOR */
```

Значение макроса **FOR**_NEST ограничивает уровень вложенности цикла. **(**По умолчанию допускается 25 уровней вложенности**)**. Переменная ftos всегда имеет значение индекса начала стека.

Для обработки стека вам понадобятся две функции fpush**()** и fpop**()**, которые приведены ниже.

```
     /* Поместить элемент в стек */
void fpush(i)
```

```
     struct for_stack i;
     {
          if(ftos > FOR_NEST)
               serror(10);
          fstack[ftos]=i;
          ftos++;
     }
     struct for_stack fpop()
     {
          ftos--;
          if(ftos < 0) serror(11);
          return(fstack[ftos]);
     }
```

Итак, после того, как вы получили возможность ознакомиться со всеми необходимыми вспомогательными функцмями, приведем полный текст функции, реализующей операторы **FOR** и NEXT.

```
     /* Простейшая реализация оператора цикла FOR */
     void exec_for()
     {
          struct for_stack i;
          int value;
          get_token(); /* чтение управляющей переменной */
          if(!isalfa(*token)) {
               serror(4);
               return;
          }
          i.var=toupper(token) - 'A'; /* сохраним индекс */
          get_token(); /* чтение символа равенства */
          if(*token != '=') {
          serror(3);
          return;
          }
          get_exp(&value); /* получить начальное значение */
          variables[i.var] = value;

          get_token();
          if(tok != TO) serror(9); /* чтение и анализ TO */
          get_exp(&i.target); /* получить конечное значение */
/* Если цикл выполняется последний раз, поместить информацию
   в стек*/
          if(value >= variables[i.var]) {
               i.loc = prog;
               fpush(i);
          }
          else { /* пропустить весь цикл целиком */
               while(tok != NEXT) get_token();
          }
        }
 /* Выполнение оператора NEXT */
     void next()
     {
         struct for_stack i;
         i = fpop(); /* чтение информации о цикле */
         variables[i.var]++; /* увеличение переменной цикла */
         if(variables[i.var] > i.target) return; /* конец */
         fpush(i); /* в противном случае запомнить информацию */
         prog = i.loc; /* цикл */
     }
```

Как именно работает эта подпрограмма, вполне ясно из комментариев к ней. Следует заметить, что анализ на возможность выхода из цикла по оператору **GOTO** в данном случае не выполняется. Поэтому использование **GOTO** в теле цикла может привести к искажению содержимого стека, что вобщем-то не желательно.

Решение проблемы реализации цикла **FOR** с помощью применения стековых структур является, в общем случае, типичным для реализации циклических конструкций. Так как интерпретатор SMALL BASIC не поддерживает другие типы циклических конструкций, то вы можете самостоятельно разработать подпрограммы реализации циклов **WHILE** и **DO-WHILE.** Как вы сможете увидеть в следующем параграфе, стековые структуры ипользуются при реализации и других конструкций программирования, допускающих вложенность.

ОПЕРАТОР GOSUB.

Хотя BASIC не поддерживает отдельные подпрограммы, но имеется возможность вызвать отдельные части программы с помощью оператора GOSUB и вернутся из нее с помощью оператора RETURN. Основной формой GOSUB-RETURN является следующая:

```
GOSUB <номер строки>
        .
        .
        .
        <номер строки>
        .
        тело подпрограммы
        .
        .
RETURN
```

Вызов подпрограммы требует использования стека. Очевидно, что это реализовать в данном случае проще, чем для оператора **FOR** потому, что каждому оператору **RETURN** соответствует свой GOSUB. Описание стека GOSUB показано ниже.

```
    char *gstack[SUB_NEST]; /* стек подпрограмм */
    int gtos; /* индекс верхней части стека  */
     Функция gosub() и вспомогательные программы приведены ниже.
    /* Простейшая реализация оператора GOSUB  */
    void gosub()
    {
      char *loc;
      get_token();
      /* поиск метки обращения */
      loc = find_label(token);
      if(loc=='\0')
        serror(7); /* метка не найдена */
      else {
        gpush(prog); /* запомним место, куда надо вернуться */
        prog = loc; /* старт программы с точки loc */
      }
    }
    /*  Возврат из подпрограммы */
    void greturn()
    {
      prog = gpop();
    }
    /* Поместить элемент в стек операторов GOSUB (подпрограмм) */
    void gpush(s)
    char *s;


    {
      gtos++;
      if(gtos == SUB_NEST) {
        serror(12);
        return;
      }
      gstack[gtos]=s;
```

```
    }
 /* Вытолкнуть элемент из стека операторов GOSUB (подпрограмм) */
    char *gpop()
    {
      if(gtos==0) {
        serror(13);
        return;
      }
      return(gstack[gtos--]);
    }
```

Команда GOSUB работает следующим образом. Текущее значение prog помещается в стек GOSUB. Затем prog присваивается адрес строки, с которой начинается подпрограмма, и начинается выполнение подпрограммы. Когда встречается **RETURN**, из стека GOSUB выталкивается очередное значение и присваивается prog. Далее выполнение продолжается со следующей строки после оператора GOSUB. Оператор GOSUB допускает вложенность.

ПОЛНЫЙ ФАЙЛ ИНТЕРПРЕТАТОРА.

Все тексты SMALL BASIC, исключая тексты синтаксического анализатора выражений, приведены ниже. Вам необходимо откомпилировать файлы интерпретатора и анализатора, а затем отредактировать их.

```
   /* Минимальный BASIC-интепретатор */
   #include "stdio.h"
   #include "setjmp.h"
   #include "math.h"
   #include "ctype.h"
   #include "stdlib.h"
   #define NUM_LAB 100
   #define LAB_LEN 10
   #define FOR_NEST 25
   #define SUB_NEST 25
   #define PROG_SIZE 10000
   #define DELIMITER 1
   #define VARIABLE  2
   #define NUMBER    3
   #define COMMAND   4
   #define STRING    5
   #define QUOTE     6
   #define PRINT 1
   #define INPUT 2
   #define IF    3
   #define THEN  4
   #define FOR   5
   #define NEXT  6
   #define TO    7
   #define GOTO  8
   #define EOL   9
   #define FINISHED 10
   #define GOSUB 11
   #define RETURN 12
   #define END   13
   char *prog; /* содержит выражение для анализа */
   jmp_buf e_buf; /* содержит сртеду для longjmp() */
   int variables[26]= {   /* 26 переменных пользователя A - Z */
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0
   };
     struct commands { /* Просмотр таблицы ключевых слов */
       char command[20];
```

```
  char tok;


} table[] = { /* Команда должна вводится прописными */
  "print",PRINT, /* буквами в эту таблицу */
  "input",INPUT,
  "if",IF,
  "then",THEN,
  "goto",GOTO,
  "for",FOR,
  "next",NEXT,
  "to",TO,
  "gosub",GOSUB,
  "return",RETURN,
  "end",END,
  "",END  /* Маркер конца таблицы */
};
char token[80];
char token_type, tok;
struct label {
  char name[LAB_LEN];
  char *p; /*      */
};
struct label label_table[NUM_LAB];
char *find_label(), *gpop();
struct for_stack {
  int var; /* переменная счетчика */
  int target; /* конечное значение */
  char *loc;
} fstack[FOR_NEST]; /* стек цикла FOR/NEXT */
struct for_stack fpop();
char *gstack[SUB_NEST]; /* стек оператора GOSUB */
int ftos; /* индекс начала стека FOR */
int gtos; /* индекс начала стека GOSUB */
void print(), scan_labels(), find_eol(),exec_goto();
void exec_if(), exec_for(), next(), fpush(), input();
void gosub(), greturn(), gpush(), label_init();
main(argc, argv)
int argc;
char *argv[];
{
  char in[80];
  int answer;
  char *p_buf;
  char *t;
  if(argc!=2) {
    printf("Используйте формат: run <filename>\n");
    exit(1);
  }


  /* Выделение памяти для программы */
  if(!(p_buf=(char *) malloc(PROG_SIZE))) {
    printf("Ошибка при выделении памяти ");
    exit(1);
  }
  /* Загрузка программы для выполнения */
  if(!load_program(p_buf,argv[1])) exit(1);
  if(setjmp(e_buf)) exit(1); /* инициализация буфера
                                нелокальных переходов */
  prog = p_buf;
  scan_labels(); /* поиск метки в программе */
  ftos = 0; /* инициализация индеса стека FOR */
  gtos = 0; /* инициализация индеса стека GOSUB */
```

```
  do {
    token_type = get_token();
    /* проверка на оператор присваивания */
    if(token_type==VARIABLE) {
      putback(); /* возврат пер. обратно во входной поток */
      assignment(); /* должен быть оператор присваивания */
    }
    else /* это команда */
      switch(tok) {
        case PRINT:
          print();
          break;
        case GOTO:
          exec_if();
          break;
        case FOR:
          exec_for();
          break;
        case NEXT:
          next();
          break;
        case INPUT:
          input();
          break;
        case GOSUB:
          gosub();
          break;
        case RETURN:
          greturn();
          break;
        case END:
          exit(0);
      }
  } while (tok != FINISHED);
}
/* Загрузка программы. */
load_program(p, fname)
char *p;


char *fname;
{
  FILE *fp;
  int i=0;
  if(!(fp=fopen(fname, "rb"))) return 0;
  i = 0;
    do {
       *p = getc(fp);
       p++; i++;
    } while(!feof(fp) && i<PROG_SIZE);
    *(p-2) = '\0'; /* символ конца программы */
    fclose(fp);
    return 1;
  }
  /* Присваивание переменной значения */
  assigment()
  {
    int var, value;
    /* Получить имя переменной */
    get_token();
      if(!isalpha(*token)) {
        serror(4); /* это не переменная */
        return;
      }
```

```
  /* вычисление индекса переменной */
  var = toupper(*token)-'A';
  /* получить знак равенства */
  get_token();
  if(*token!='=') {
    serror(3);
    return;
  }
  /* получить значение, присваемое переменной */
  get_exp(&value);
  /* присвоить это значение */
  variables[var] = value;
}
/* Простейшая реализация оператора PRINT */
void print()
{
  int answer;
  int len=0, spaces;
  char last_delim;
  do {
    get_token(); /* получить следующий элемент списка  */
    if(tok==EOL || tok==FINISHED) break;


    if(token_type==QUOTE) { /* это строка */
      print(token);
      len += strlen(token);
      get_token();
    }
    else { /* это выражение */
      putback();
      get_exp(&answer);
      get_token();
      len +=printf("%d", answer);
    }
    last_delim = *token;
    if(*token==';') {
/* вычисление числа пробелов при переходе к следующей табуляции*/
      spaces= 8- (len % 8);
      len += spaces; /* включая позицию табуляции  */
      while(spaces) {
        print(" ");
        spaces--;
      }
    }
    else if(*token==','); /* ничего не делать */
    else if(tok!=EOL && tok!=FINISHED)  serror(0);
  } while (*token == ';' || *token == ',');
 if(tok==EOL || tok==FINISHED) {
 if(last_delim != ';' && last_delim != ',') printf("\n");
 }
  else serror(0); /* отсутствует ',' или ';'  */
}
/* Поиск всех меток */
void scan_labels()
{
  int addr;
  char *temp;
label_init();  /* обнуление всех меток */
temp = prog; /* сохраним указатель на начало программы*/
  /* Если первая лексема файла есть метка  */
  get_token();
  if(token_type==NUMBER) {
    strcpy(label_table[0].name,token);
```

```
        label_table[0].p=prog;
      }
      find_eol();
      do {
        get_token();
        if(token_type==NUMBER) {
          addr =get_next_label(token);
          if(addr==-1 || addr==-2) {
              (addr==-1) ?serror(5):serror(6);
          }


          strcpy(label_table[addr].name, token);
label_table[addr].p = prog; /* текущий указатель программы */
        }
        /* если строка не помечена, то поиск следующей */
        if(tok!=EOL) find_eol();
      } while(tok!=FINISHED);
      prog = temp; /* сохраним оригинал */
    }
    /* Поиск начала следующей строки */
    void find_eol()
    {
      while(*prog!='\n'  && *prog!='\0') ++prog;
      if(*prog) prog++;
    }
    /* Возвращает индекс ена следующую свободную позицию
       массива меток. -1, если массив переполнен.
                      -2, если дублирование меток. */
    get_next_label(s)
    char *s;
    {
      register int t;
      for(t=0;t<NUM_LAB;++t) {
        if(label_table[t].name[0]==0) return t;
     if(!strcmp(label_table[t].name,s)) return -2; /*дубль*/
      }
      return -1;
    }
/* Поиск строки по известной метке. Значение 0 возвращается,
   если метка не найдена; в противном случае возвращается
   указатель на помеченную строку программы        */
    char *find_label(s)
    char *s;
    {
      register int t;
      for(t=0; t<NUM_LAB; ++t)
  if(!strcmp(label_table[t].name,s)) return label_table[t].p;
      return '\0'; /* состояние ошибки */
    }
    /* Реализация оператора GOTO */
    void exec_goto()
    {
      char *loc;
      get_token(); /* получить метку перехода */


      /* Поиск местоположения метки */
      loc = find_label(token);
      if(loc=='\0')
        serror(7);   /* метка не обнаружена  */
      else prog=loc; /* старт программы с указанной точки  */
    }
 /* Инициализация массива хранения меток. По договоренности
```

```
нулевое значение метки символизирует пустую ячейку массива *
   void label_init()
   {
   register int t;
   for(t=0; t<NUM_LAB; ++t) label_table[t].name[0]='\0';
 }
   /* Реализация оператора IF  */
   void exec_if()
   {
     int x , y, cond;
     char op;
     get_exp(&x); /* получить левое выражение */
     get_token(); /* получить оператор */
     if(!strchr("=<>", *token)) {
     serror(0);      /* недопустимый оператор */
     return;
   }
   op=*token;
   get_exp(&y);  /* получить правое выражение */
   /* Определение результата */
   cond=0;
   switch(op) {
       case '=':
     if(x==y) cond=1;
     break;
       case '<':
     if(x<y) cond=1;
     break;
       case '>':
     if(x>y) cond=1;
     break;
   }
   if(cond) {  /* если значение IF "истина"  */
     get_token();
     if(tok!=THEN) {
        serror(8);
        return;
    } /* иначе, программа выполняется со следующей строки */
   }
   else find_eol(); /* поиск точки старта программы */
 }
 /* Реализация цикла FOR */
 void exec_for()


 {
   struct for_stack i;
   int value;
   get_token(); /* получить управляющую переменную */
   if(!isalpha(*token)); {
     serror(4);
     return;
   }
   i.var=toupper(*token)-'A'; /* сохранить ее индекс */
   get_token(); /* получить знак равенства */
   if(*token!='=') {
     serror(3);
     return;
   }
   get_exp(&value); /* получить начальное значение  */
   variables[i.var]=value;
   get_token();
   if(tok!=TO) serror(9); /* если нее нашли TO */
   get_exp(&i.target); /* получить конечное значение */
```

```
      /* Если цикл выполняется последний раз, поместить
         информацию в стек  */
      if(value>=variables[i.var]) {
        i.loc = prog;
        fpush(i);
      }
      else /* пропустить весь цикл */
        while(tok!=NEXT) get_token();
    }
    /* Реализация оператора NEXT */
    void next()
    {
      struct for_stack i;
      i = fpop(); /* чтение информации о цикле */
     variables[i.var]++; /* увеличение управляющей переменной*/
      if(variables[i.var]>i.target) return; /* конец цикла */
      fpush(i); /* иначе, сохранить информацию в стеке */
      prog = i.loc; /* цикл */
    }
    /* Поместить информацию в стек FOR */
    void fpush(i)
    struct for_stack i;
    {
      if(ftos>FOR_NEST)
      serror(10);
    fstack[ftos]=i;
    ftos++;
  }


  struct for_stack fpop()
  {
    ftos--;
    if(ftos<0) serror(11);
    return(fstack[ftos]);
  }
  /* Реализация оператора INPUT */
  void input()
  {
    char str[80], var;
    int i;
get_token(); /*просматривается если существует строка символов*/
    if(token_type==QUOTE) {
   printf(token); /* если да, то ее печать и контроль ',' */
      get_token();
      if(*token!=',') serror(1);
      get_token();
    }
    else printf("? "); /* выдача строки по умолчанию */
 var = toupper(*token)-'A'; /* получить индекс имени переменной*
    scanf("%d",&i);   /* чтение ввода данных */
    variables[var] = i;  /* сохранение данных */
  }
  /* Реализация оператора GOSUB */
  void gosub()
  {
    char *loc;
    get_token();
    /* поиск метки вызова */
    loc = find_label(token);
    if(loc=='\0')
      serror(7); /* метка не определена */
    else {
      gpush(prog); /* запомним место, куда вернемся */
```

```
      prog = loc; /* старт программы с указанной точки */
    }
  }
  /* Возврат из подпрограммы, вызвынной по GOSUB */
  void greturn()
  {
    prog = gpop();
  }
  /* Помещает данные в стек GOSUB */
  void gpush(s)
  char *s;
  {
    gtos++;
    if(gtos==SUB_NEST) {


      serror(12);
      return;
    }
    gstack[gtos]=s;
  }
  /*   */
  char *gpop()
  {
    if(gtos==0) {
      serror(13);
      return;
    }
    return(gstack[gtos--]);
  }
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ИНТЕРПРЕТАТОРА SMALL BASIC

Теперь вы можете запускать простейшие BASIC-программы.

```
PRINT "Эта программа демонстрирует все команды."
FOR X = 1 TO 100
PRINT X, X/2; X, X*X
NEXT
GOSUB 300
PRINT "Привет"
INPUT H
IF H<11 THEN GOTO 200
PRINT 12-4/2
PRINT 100
200 A = 100/2
IF A>10 THEN PRINT "Все нормально"
PRINT A
PRINT A+34
INPUT H
PRINT H
INPUT "Это тест",y
PRINT H+Y
END
300 PRINT "Это подпрограмма"
    RETURN
PRINT "Эта подпрограмма демонстрирует вложенный GOSUB"
INPUT "Введите число: ", i
GOSUB 100
END
100 FOR T = 1 TO I
  X = X+I
  GOSUB 150
NEXT
```

```
RETURN
150 PRINT X;
    RETURN
print "Эта подпрограмма вычисляет объем куба "
input "Введите длину парвой стороны: ", l
input "Введите длину второй стороны: ", w
input "Введите длину третей стороны: ", d
t = l * w * d
print "Обьем равен: ", t


PRINT "Эта программа демонстрирует вложенные циклы"
FOR X = 1 TO 100
 FOR Y = 1 TO 10
   PRINT X; Y; X*Y
 NEXT
NEXT
```

РАСШИРЕНЕИЕ ВОЗМОЖНОСТЕЙ ИНТЕРПРЕТАТОРА

Основным пунктом расширения функций интерпретатора и их модификации является ограниченность его лишь возможностью интерпретировать язык BASIC. Основная масса технических приемов, описанных в этой главе, может использовать при создании интерпретаторов для различных процедурных языков программирования. Потому вы, в принципе, можете разработать свой язык, учитывающий ваш взгляд на программирование и ваш стиль программирования.

Дополнительные команды и соответствующие им основные форматы лексем описаны в этой главе. Для использования различных дополнительных типов переменных вы должны использовать массивы структур для хранения таких переменных; одно из полей этой структуры должно индицировать тип переменной, а остальные поля предназначены для хранения значений этих переменных. Для использования строчных переменных вам необходимо установить таблицу строк. Простейшим путем реализации строк фиксированной длины является использование фиксированных участков памяти в 255 байт для размешения строк.

И последняя мысль: типы операторов, которые вы можете интерпретировать, ограничены лишь вашей фантазией.

О МАНИПУЛИРОВАНИИ ЭКРАНОМ И ГЕНЕРАЦИИ ЗВУКА

На всем протяжении этой книги мы касались, в основном, тех аспектов программирования на Си, которые могут заинтересовать профессиональных программистов, чье основное занятие программирование на Си. Так как возможности программы по взаимодействию с пользователем часто ограничиваются возможностями, представленными в рамках пользовательского интерфейса, то в этой главе содержатся завершенные сведения, так сказать, окончательная точка зрения на возможность и целесообразность манипулирования с экраном дисплея при разработке пользовательского интерфейса. Основное внимание в этой главе уделяется вопросам отображения различных фрагментов текста в разных цветах. Дополнительно в этой главе рассматриваются некоторые другие вопросы программирования пользовательского интерфейса такие, как изменение размера и формы курсора, скроллинг (прокрутка) части текста, сохранение содержимого экрана в виде дискового файла. Использование со вкусом звуковых возможностей компьютера позволяет в значительной мере "оживить" работу пользователя с вашей программой, а также акцентировать внимание пользователя на ряде моментов и ситуаций, возникающих во

время работы. В связи с этим в главу включен параграф, поясняющий возможности пользователя по генерации звуков различных частот и созданию различных звуковых эффектов с использованием динамика компьютера.

Подпрограммы, описанные в этой главе, являются машинно-зависимыми. Они могут функционировать на IBM PC, XT, AT, PS/2 и совместимых с ними моделях. Большинство из подпрограмм требуют наличия в вашей конфигурации компьютера цветного дисплея (адаптера). Если вы имеете несовместимый с вышеперечисленными моделями компьютер, то вам необходимо будет внести в подпрограммы соответствующие изменения.

ИСПОЛЬЗОВАНИЕ ЦВЕТА В ТЕКСТОВОМ РЕЖИМЕ

Ранее вы могли видеть великолепные, профессионально написанные программы, которые не используют цветовые возможности. Как вы уже уяснили из Глава VIIIы 1, семейство компьютеров PC поддерживает различные видеорежимы. Если вы имеете в своей системе цветной адаптер, режим работы которого по умолчанию установлен равным 3, то это означает, что специфицирован цветной режим отображения текста 80*25 строк. По умолчанию текст отображается на экране в белом цвете, однако, имеется возможность отображать текст в других цветах.

Атрибутный байт текстового режима.

Каждый символ отображается на экране дисплея в соответствии с его атрибутным байтом, определяющим как именно отображается символ (см. главу 1). Если компьютер включает в себя цветной адаптер, работающий в видеорежиме, определяемом значением 3, то соответствующее значение атрибутного байта определяет цвет отображаемого символа, цвет фона, интенсивность отображения символа (уровень яркости), а также устанавливает или отменяет режим мерцания символа. Состав атрибутного байта показан в Таблице 8.1.

Биты 0, 1 и 2 атрибутного байта определяют компоненты цвета символа. Например, если установлен бит 0 (значение бита равно 1), то символ отображается в голубом цвете. Если значение всех этих битов не установлено, то символ является неотображаемым. Запомните, что цвета накладываются друг на друга. Если значения всех этих битов установлены (равны 1), символ отображается в белом цвете. Если вы установили значения двух из этих битов, то будет генерирован либо ярко-красный, либо голубой (циановый) цвет символа. Биты с 4 по 6 используются для установки цвета фона. Если значение этих битов не установлено (равно 0), то цвет фона будет черным, в противном случае цвет фона определяется в соответствии со специфицированным значением битов.

На заре микрокомпьютеров режимом, в котором видеосистемой отображались символы по умолчанию, был режим полной яркости, однако наряду с этим режимом имелась возможность отображать символы в режиме пониженной яркости. После реализации IBM PC пользователю был предложен альтернативный путь: По умолчанию отображение символов видеосистемой PC выполняется в режиме "нормальной" яркости, но вы имеете возможность отображать символы в режиме повышенной яркости, устанавливая значение 1 для соответствующего бита атрибутного байта (бита повышенной яркости) В добавок ко всему вы можете установить режим мерцания символа, установив значение соответствующего бита.

Таблица 8-1.

Состав атрибутного байта при работе в 3 видеорежиме

Бит	Устанавливаемое значение
0	Голубой цвет символа

1	Зеленый цвет символа
2	Красный цвет символа
3	Повышенная яркость символа
4	Голубой цвет фона
5	Зеленый цвет фона
6	Красный цвет фона
7	Мерцание символа

В предыдущих параграфах были рассмотрены функции, которые выполняли считывание символов на экран, используя при этом как обращение к BIOS, так и непосредственный доступ к видеопамяти. Непосредственный доступ к видеопамяти является необходимым условием повышения скорости реакции задач на действия пользователя. Однако непосредственный доступ к видеопамяти значительно снижает возможность переносимости программ (их мобильность), а также является серьезной помехой при использовании программ в мультизадачных операционных системах типа OS/2. Функции, рассматриваемые в этой главе, используют возможности BIOS и видеопамяти по той причине, что сами по себе эти функции более мобильны и, в конечном итоге, для ускорения их быстродействия обычно не требуется стандартный вывод на дисплей. Следует отметить, что интуитивно использование непосредственного доступа к видеопамяти является более предпочтительным.

Отображение строки в определенном цвете.

Отображение строки в определенном цвете не является столь трудной задачей, как вам может казаться на первый взгляд, если вы используете функции записи символа, которые используют, в свою очередь, возможности BIOS и видеопамяти (ROM-BIOS). ROM-BIOS прерывание 10H, функция 9 позволяет отобразить текущий символ (один!) в позиции курсора и его атрибуты. Проблема состоит лишь в перемещении курсора по записываемой вами строке, но это должна осуществлять непосредственно ваша подпрограмма.

В соответствии с этим возникает, во-первых, необходимость определения текущей позиции курсора. Для этого используется функция read_cursor_xy(), представленная ниже. Эта функция использует ROM-BIOS-прерывание 10H, функцию 3, для чтения текущих координат позиции курсора X и Y. Координаты позиции курсора возвращаются в качестве значений аргументов функции.

```
     /* Чтение текущих координат позиции курсора */
void read_cursor_xy(x,y)
char *x,*y;
{
     union REGS r;
     r.h.ah = 3;  /* чтение текущей позиции курсора */
     r.h.bh = 0;  /* видеостраница */
     int86(0x10,&r,&r);
     *y = r.h.dl;
     *x = r.h.dh;
}
```

После определения координат текущей позиции курсора, функция, которая выполняет печать строки, должна осуществить перемещение курсора к следующему символу, с тем, чтобы используя ROM-BIOS-прерывание напечатать его. Для перемещения курсора целесообразно использовать функцию goto_xy(), которая была уже рассмотрена ранее и приводится в этой главе для полноты изложения материала.

```
     /* Перемещение курсора в позицию, специфицированную
        координатами X и Y
     */
void goto_xy (x,y)
int x,y;
```

```
{
       union REGS r;
       r.h.ah = 2; /* функция адресации курсора */
       r.h.dl = x; /* координата столбца */
       r.h.dh = y; /* координата строки */
       r.h.bh = 0; /* видеостраница */
       int86(0x10,&r,&r);
}
```

Функция color_puts(), представленная ниже, отображает специфицированную пользователем строку в указанном цвете.

```
     /* Печать строки в цвете   */
void color_puts(s,color)
char *s;  /* строка */
char color; /* цвет строки */
{
      union REGS r;
      char x,y;
      read_cursor_xy(&x,&y); /* получение текущей позиции курсора
                     */
      while (*s) {
      if(*s == '\n') {  /* обработка символа новой строки */
          printf("\n");
          s++;
          x = 0; y++;  /* переход на следующую строку */
          continue;
 }
    r.h.ah = 9; /* функция отображения символа и его атрибутов */
    r.h.al = *s++;    /* отображаемый символ */
    r.h.bl = color;   /* атрибуты цвета */
    r.h.bh = 0;       /*видеостраница 0 */
    r.x.cx = 1; /* отобразить за единицу времени ( такт ) */
     int86(0x10,&r,&r);
     x++;
     goto_xy(x,y); /* перемещение курсора */
 }
}
```

Как вы можете видеть, отображаемый символ запоминается в регистре AL, атрибуты цвета символа - в регистре BL, номер видеостраницы - в регистре BH, а количество интервалов времени (тактов процессора), за которое будет отображен символ - в регистре CX. Заметим, что функция также обрабатывает специальный символ новой строки ('\n'). Вы можете также, по желанию, организовать обработку символов табуляции ('\t'), двойных кавычек (") и других специальных символов.

Использование функции color_puts() предполагает наличие ряда макроопределений в начале вызывающей функцию программы. Перечень макроопределений представлен ниже

```
       #define BLUE             1
       #define GREEN        2
       #define RED             4
       #define INTENSE      8
       #define BLUE_BACK    16
       #define GREEN_BACK   32
       #define RED_BACK     64
       #define BLINK        128
```

Используя эти макросы, вы можете по своему усмотрению выдать на экран строку текста на фоне установленного вами цвета, а также саму строку в определенном вами цвете. Вы можете также управлять режимом отображения строки (повышенная яркость или мерцание). Комбинируя цвета, режимы мерцания или повышенной яркости для одного или совокупности символов, вы можете добиться любого желаемого вами эффекта. Например, представленная ниже строка

программы приведет к отображению строки "А это - текст" в режиме повышенной яркости в голубом (циановом) цвете:

```
color_puts("А это - текст",GREEN | RED | INTENSE );
```

Использование цвета.

Довольно эффективно использование многоцветного текста в различных приложениях. Во всяком случае многоцветный текст смотрится всегда намного эстетичней, чем монохромный. Однако при использовании многоцветного текста необходимо придерживаться следующих основных положений:

- Избегайте использования "нестандартных" цветов. Наиболее общим лучшим вариантом является отображение белых с имволов на черном фоне. Предпочтительнее вместо отображения какой-то важной информации в контрастном цвете отображать ее в режиме повышенной яркости.
- Наиболее эффективным признано использование цветных рамок экрана и окон.
- В ряде ситуаций полезным оказывается отображение отрицательного остатка (например, денежной суммы) в красном цвете.
- Отображение текущей строки (или части текущей строки) в контрастном цвете является, пожалуй, лучшим приемом индикации положения действий пользователя на экране в текущий момент времени.

ИЗМЕНЕНИЕ РАЗМЕРА КУРСОРА

Большинство пользователей даже не представляют насколько велики возможности семейства машин IBM PC. В частности, они позволяют изменять размер курсора. По умолчанию курсор отображается в виде одной мерцающей строчки (развертки дисплея). Однако пользователь может варьировать размером курсора от одной строки развертки дисплея до полного размера (высоты) символа. В цветном текстовом режиме курсор может иметь высоту от 0 до 8 строк развертки. (В монохромном режиме курсор может иметь высоту от 0 до 14 строк развертки, однако в данном параграфе мы будем рассматривать только цветной режим). Нижняя строка развертки имеет номер 0. Лучше всего рассматривать изменение курсора именно относительно строки развертки 0, так как применение другого метода может привести к значительным расхождениям результатов на различных компьютерах. (В принципе применение других методов возможно, однако вам необходимо помнить, что при их использовании вы можете не добиться соответствия форм курсора при решении одной и той же задачи на разных компьютерах). При условии, что вы будете рассматривать изменение формы курсора относительно нулевой строки развертки, изменение формы курсора будет выполняться аналогично изображенному на рис.8-1.

Для установления размера курсора вам необходимо использовать ROM-BIOS-прерывание 10H, функцию 1, которая устанавливает размер курсора. Начало курсора (начальная строка развертки) - запоминается в регистре CH, а конец (конечная строка развертки) в регистре CL.

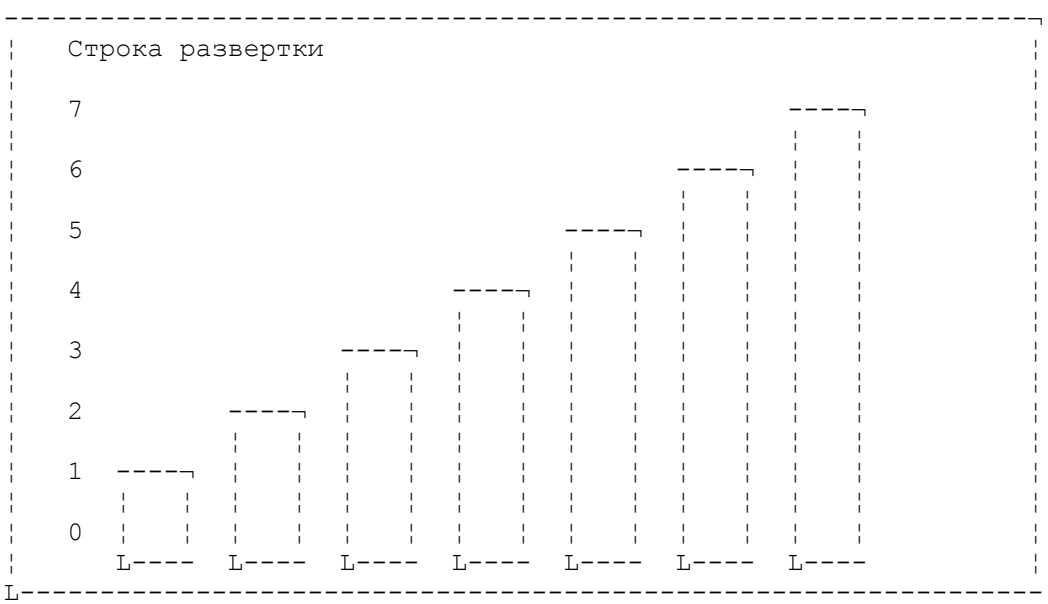


Рис. 8-1. Возможность изменения формы курсора в цветном режиме.

Функция size_cursor(), представленная ниже, устанавливает размер курсора.

```
     /* Установление размера курсора */
void size_cursor(start,end)
char start,end; /* начальная и конечная строки развертки */
{
     union REGS r;
     r.h.ah = 1; /* функция адресации курсора */
     r.h.ch = start;
     r.h.cl = end;
     int86(0x10,&r,&r);
}
```

При использовании функции size_cursor() укажите желаемые начальную и конечную строки развертки, определяющие размер курсора. Например, следующая конструкция позволяет установить высоту курсора в три строки развертки:

```
size_cursor(0,2);
```

Форма курсора может быть изменена либо очередным вызовом функции size_cursor(), либо изменением видеорежима.

Использование курсоров различной формы позволяет не только полностью удовлетворить ваши эстетические требования, но и повысит наглядность программы. Имейте в виду, что большой мерцающий курсор вызывает у пользователей раздражение.

СКРОЛЛИНГ ЧАСТИ ЭКРАНА

Две совместно используемые функции ROM-BIOS-прерывания позволяют осуществлять скроллинг вперед и назад части экрана. Эти функции были включены в ROM-BIOS для поддержки многооконных интерфейсов. Как вы знаете, когда курсор расположен в двадцать пятой строке и вы нажали клавишу <ВВОД>, то автоматически осуществляется перемещение текста на одну строку вверх с целью отображения новой строки в нижней части экрана. Точно так же, с помощью функций 6 и 7 прерывания ROM-BIOS 10H, можно осуществить скроллинг лишь части экрана. Функция 6 позволяет выполнить скроллинг в окне вниз (вперед), а функция 7 - вверх (назад).

Обе функции при вызове используют информацию, хранимую в определенных регистрах. Занесите количество строк, на которые будет "прокручиваться" текст (мощность скроллинга) в регистр AL. Номер верхней левой строки, ограничивающей ваше "окно", занесите в регистр CH, а номер верхнего левого столбца - в регистр CL.

Номер нижней левой строки занесите в регистр DH, а номер нижнего правого столбца - в регистр DL. В конце запомните в регистре BH атрибуты режима отображения, которые будут определять, как именно будут отображаться в процессе скроллинга новые строки. Функция scroll_window() представлена ниже.

```
     /* Скроллинг в окне вперед и назад */
void scroll_window(startx,starty,endx,endy,lines,direct)
char startx,starty;/* верхний левый угол */
char endx,endy;    /* нижний правый угол */
char lines;        /* число строк прокрутки */
char direct;       /* вперед или назад */
{
     union REGS r;
     if( direct == UP ) r.h.ah = 6; /* скроллинг вперед */
     else r.h.ah = 7; /* скроллинг вниз (назад) */
     r.h.al = lines;
     r.h.ch = starty;
     r.h.cl = startx;
     r.h.dh = endy;
     r.h.dl = endx;
     r.h.bh = 0;     /* режим отображения */
     int86(0x10,&r,&r);
}
```

Вы можете определить макрос UP как имеющий некоторое значение. Вы также можете определить макрос DOWN, значение которого будет отлично от UP, а затем использовать при необходимости осуществления скроллинга в окне эти макроопределения. Такой прием значительно упростит вашу программу. Функция scroll_window() присваивает регистру BH значение 0 для сохранения пустых строк, однако вы можете изменить это значение по своему усмотрению.

ПРОСТЕЙШАЯ ДЕМОНСТРАЦИОННАЯ ПРОГРАММА

Эта программа позволяет продемонстрировать работу функций, рассмотренных в данной главе до настоящего момента. Эти функции позволяют изменить форму курсора, выдать сообщение в цвете, а также использовать скроллинг части экрана. Результат работы этой программы представлен на рис.8-2.

```
а)     Это - тест
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
  ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
  ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
  eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
  fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
  ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
  hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
  iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
  jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj
  kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk
  lllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll
  mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
  nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
  oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
  ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
  qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
  rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
  sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
  ttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
  uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
```

```
¦     vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv    ¦
¦                                                               ¦
¦     б) Это - тест.                                            ¦
¦     aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa    ¦
¦     bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb    ¦
¦     cccccccccccccccccccccccccccccccccccccccccccccccccccccc    ¦
¦     dddddddddddddddddddddddddddddddddddddddddddddddddddddd    ¦
¦     eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee    ¦
¦     ffffffffffffffffffffffffffffffffffffffffffffffffffffff    ¦
¦     gggggggggggggggggggggggggggggggggggggggggggggggggggggg    ¦
¦     hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh    ¦
¦     iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii    ¦
¦     jjjjjjjjjj                     jjjjjjjjjjjjjjjjjjjjjjj    ¦
¦     kkkkkkkkkk                     kkkkkkkkkkkkkkkkkkkkkkk    ¦
¦     llllllllll                     lllllllllllllllllllllll    ¦
¦     mmmmmmmmmmjjjjjjjjjjjjjjjjjjjjjmmmmmmmmmmmmmmmmmmmmmmm    ¦
¦     nnnnnnnnnnkkkkkkkkkkkkkkkkkkkkknnnnnnnnnnnnnnnnnnnnnnn    ¦
¦     oooooooooolllllllllllllllllllllooooooooooooooooooooooo    ¦
¦     pppppppppppppppppppppppppppppppppppppppppppppppppppppp    ¦
¦     qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq    ¦

¦     rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr    ¦
¦     ssssssssssssssssssssssssssssssssssssssssssssssssssssss    ¦
¦     tttttttttttttttttttttttttttttttttttttttttttttttttttttt    ¦
¦     uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu    ¦
¦     vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv    ¦
¦                                                               ¦
L----------------------------------------------------------------
```

Рис. 8-2 Результат работы программы.

```
/* Демонстрационная программа печати текста в цвете,
   изменения формы курсора и скроллинга в окне
*/
  #include "dos.h"
  #define BLUE        1
  #define GREEN       2
  #define RED         4
  #define INTENSE     8
  #define BLUE_BACK   16
  #define GREEN_BACK  32
  #define RED_BACK    64
  #define BLINK      128
     #define UP       0
  #define DOWN        1
  void mode(),color_puts(),palette(), read_cursor_xy();
  void goto_xy(),size_cursor(),scroll_window();
     main()
     {
      int i,j;
   mode(3); /* режим текстовый, цветной */
      size_cursor(0,3);
      goto_xy(0,0);
      color_puts(" Это - тест\n",BLUE |RED | INTENSE);
      for ( i=0;i<22;i++) {
          for ( j=0;j<79;j++)
          printf("%c",i+'a');
          printf("\n");
     }
     getche();
     scroll_window(10,10,50,15,3,DOWN);
     getche();
}
/* Печать строки в цвете */
     void color_puts(s,color)
     char *s;      /* строка */
```

```
     char color;   /* цвет строки */
     {
          union REGS r;
          char x,y;
          read_cursor_xy(&x,&y); /* чтение текущей позиции
                                    курсора   */
          while (*s) {
           if(*s == '\n') { /* обработка символа новой
                               строки */
           printf("\n");
           s++;
           x = 0; y++;  /* переход к новой строке */
           continue;
          }
          r.h.ah = 9;/* отображение символов по атрибутам */
          r.h.al = *s++; /* выдаваемый символ */
          r.h.bl = color;  /* атрибут цвета */
          r.h.bh = 0; /*     видеостраница 0 */
          r.x.cx = 1; /* выдача символа за один такт */
          int86(0x10,&r,&r);
          x++;
          goto_xy(x,y);      /* перемещение курсора */
          }
     }
/* чтение текущей позиции курсора */
     void read_cursor_xy(x,y)
     char *x,*y;
     {
      union REGS r;
      r.h.ah = 3; /* чтение позиции курсора */
      r.h.bh = 0; /* видеостраница 0 */
      int86(0x10,&r,&r);
      *y = r.h.dl;
      *x = r.h.dh;
     }
/* установка палитры */
     void palette(pnum)
     int pnum;
     {
      union REGS r;
      r.h.bh = 1;  /* код для графического режима 4 */
      r.h.bl = pnum;
      r.h.ah = 11; /* функция установки палитры */
      int86(0x10,&r,&r);
     }
/* установка видеорежима */
     void mode(mode_code)
     int mode_code;

     {
      union REGS r;
      r.h.al = mode_code;
      r.h.ah = 0;
      int86(0x10,&r,&r);
     }
/* перемещение курсора в позицию x,y */
     void goto_xy(x,y)
     int x,y;
     {
      union REGS r;
      r.h.ah = 2; /* функция адресации курсора */
      r.h.dl = x; /* координаты столбца */
      r.h.dh = y; /* координаты строки */
```

```
        r.h.bh = 0; /* видеостраница 0 */
        int86(0x10,&r,&r);
       }
  /* установка размера (формы) курсора */
       void size_cursor(start,end)
       char start,end; /* начальная и конечная строки
                  развертки */
       {
        union REGS r;
        r.h.ah = 1; /* функция адресации курсора */
        r.h.ch = start;
        r.h.cl = end;
        int86(0x10,&r,&r);
       }
  /* скроллинг в окне вперед и назад */
 void scroll_window(startx,starty,endx,endy,lines,direct)
 char startx,starty; /* верхний левый угол */
 char endx,endy;     /* нижний правый угол */
 char lines;         /* мощность скроллинга */
 char direct;        /* вверх или вниз */
  {
   union REGS r;
   if( direct == UP ) r.h.ah = 6; /* скроллинг вверх */
    else r.h.ah = 7; /* скроллинг вниз */
  r.h.al = lines;
  r.h.ch = starty;
  r.h.cl = startx;
  r.h.dh = endy;
  r.h.dl = endx;
  r.h.bh = 0; /* режим отображения */
  int86(0x10,&r,&r);
  }
```

СОХРАНЕНИЕ КОПИИ ЭКРАНА В ДИСКОВОМ ФАЙЛЕ

Ни в DOS, ни в OS/2 нет утилит, сходных с утилитой печати копии экрана, позволяющей сохранять текущее содержимое экрана дисплея в дисковом файле. В этом разделе вы найдете пояснение, как можно создать программу, которая выполняла бы именно эту функцию.

Эта программа использует ROM-BIOS-прерывание 10H, функцию 8 для чтения символа из текущей позиции курсора, после чего этот символ записывается в файл на диске. Как и в предыдущем разделе, вы опять встретитесь с функцией **goto_xy()**, которая в этом случае используется для перемещения курсора последовательно по всем строкам экрана, начиная с левого верхнего угла экрана до правого нижнего угла.

Имя файла, в котором будет храниться копия экрана, указываетя в качестве аргумента программы. Если, к примеру, вы назовете свою программу, копирующую экран на диск, screen, то представленная ниже командная строка приведет к созданию копии экрана в файле с именем scr.sav:

```
C> screen scr.sav
```

А вот исходный текст самой программы копирования:

```
       /* Эта программа копирует содержимое экрана вашего
          дисплея в файл, имя которого указано в командной
          строке
       */
  #include "dos.h"
  #include "stdio.h"
  void save_screen(),goto_xy();
  main(argc,argv)
  int argc;
```

```
char *argv[];
{
  if( argc != 2 ) {
     printf(" используйте формат : screen <имя файла>");
     exit(1);
  }
  save_screen(argv[1]);
}
/* сохранение содержимого экрана в дисковом файле */
     void save_screen(fname)
     char *fname;
{
 FILE *fp;
 union REGS r;

 register char x,y;
 if( !( fp=fopen(fname,"w"))) {
     printf(" Файл не может быть открыт ");
     exit(1);
 }
 for (y=0;y<25;y++)
     for (x=0;x<80;x++) {
     goto_xy(x,y);
     r.h.ah = 8; /* чтение символа */
     r.h.bh = 0; /* видеостраница */
     int86(0x10,&r,&r);
     putc(r.h.al,fp); /* выдача (печать) символа */
 }
fclose(fp);
}
/* Перемещение курсора в позицию (x,y) */
void goto_xy(x,y)
int x,y;
{
  union REGS r;
  r.h.ah = 2; /* функция адресации курсора */
  r.h.dl = x; /* координата столбца */
  r.h.dh = y; /* координата строки */
  r.h.bh = 0; /* видеостраница */
  int86(0x10,&r,&r);
}
```

Создаваемый файл представляет собой стандартный ASCII файл, который может быть отредактирован и распечатан как обычный текстовый файл. Программа позволяет записывать лишь символы, отображенные на экране, но не позволяет сохранить соответствующие атрибуты отображения символов. Однако дополнить программу для того, чтобы она записывала и атрибуты символов несложно, и вы при желании можете это сделать самостоятельно.

А ТЕПЕРЬ ДОБАВИМ ЗВУК.

Использование со вкусом звука значительно повышает привлекательность программ. Звук может быть использован в широком спектре от тоненького "писка" машины до исполнения музыки или различных специальных эффектов. В этом параграфе вы научитесь, как можно управлять высотой и продолжительностью звучания нот, генерируемых динамиком компьютера. Мы также продемонстрируем вам некоторые наиболее интересные звуковые эффекты.

Программируемый таймер 8253.

Генерация звуков в компьютере PC выполняется с помощью программируемого таймера 8253, который применяется для управления колебаниями динамика. Управление колебаниями динамика

определяется частотой, которая, в свою очередь, определяется содержимым различных внутренних регистров. Значения этих регистров устанавливаются при записи в определенные порты. Порт 66 используется для спецификации счетчика, который использует таймер при определении интервала колебаний динамика. Таймер работает в строгом соответствии с частотой системного таймера и специфицированным значением счетчика, определяющим колебания динамика. Затем, после обнуления счетчика происходит установка нового значения счетчика, и весь цикл функционирования программируемого таймера повторяется сначала. Значение счетчика определяется по следующей формуле:

count = 1,193,180/требуемая частота

где 1,193,180 есть тактовая частота системного таймера.

Регистр-счетчик таймера 8253 устанавливается в следующей последовательности (значение счетчика задается двухбайтным числом):

1. Выдать в порт 67 значение 182 (означающее, что будет устанавливаться счетчик).
2. Выдать в порт 66 младший байт числа, определяющего значение счетчика.
3. Выдать в порт 66 старший байт числа, определяющего значение счетчика.

Динамики большинства компьютеров класса PC не позволяют воспроизводить полный спектр частот, воспринимаемых человеческим слухом (от 20 Гц до 18.000 Гц). Однако динамик позволяет воспроизводить ноты лучше, чем динамики других компьютеров в пределах 12000 Гц и даже выше. В основном же динамик используется в пределах 100-5000 Гц.

Итак, таймер установлен. Однако динамик еще не будет воспроизводить звук, так как не включен. Таймер 8253 активен постоянно, а динамик требует дополнительной команды включения. Активизация динамика осуществляется путем установки значений битов 0 и 1 регистра программируемого периферийного интерфейса, задание значений которого выполняется через порт 97. Если значения этих двух битов установлены (равны 1), то динамик издает звук частотой, установленной счетчиком 8253. Если значения этих битов равны 0, то никакой звук генерироваться не будет. Остальные биты этого байта используются другими устройствами, поэтому интерпретация значения левых битов не может быть изменена. Таким образом, для установки значений управляющих динамиком бит

необходимо выполнить следующую последовательность действий:

1. Получить текущее значение регистра из порта 97.
2. Сравнить это значение с 3 или установить равным 3.
3. Записать результат в порт 97.

Для того, чтобы выключить динамик, необходимо переслать в порт значение 253.

Простейшим приемом, позволяющим читать и писать байт из или в порт, в Си является использование соответствующих функций. В Турбо Си - это функции inportb() и outportb(). В Microsoft Си - это функции inp() и outp(). Они имеют следующий общий формат:

```
int inportb(int port);
void outportb(int port, char value);
int inp(unsigned port);
int outp(unsigned port, int value);
```

В других компиляторах Си эти функции могут иметь иные названия, но обязательно будут присутствовать в вашей библиотеке, так как являются одними из базовых функций версий Си для ПЭВМ. В программах, приведенных в этом параграфе, используются функции Турбо Си.

Простейший способ проверки слуха.

Вы обладаете возможностью сделать несколько грубый, но

эффективный тест слуха, который в состоянии обнаружить некоторые типы дефекта слуха. Как вы ранее узнали, динамик большинства компьютеров серии PC не воспроизводит звуки выше 12000 Гц. Однако ряд людей, у которых отмечены некоторые отклонения слуха, не могут услышать звук даже такой частоты. Фактически, тестируя свой слух, вы будете несколько удивлены тем, насколько высоким окажется звук с частотой 12000 Гц. (Предупреждение: тестирование слуха с помощью этого теста можно производить лишь ради шутки. Он, естественно, не позволяет действительно оценить слух испытуемого. Поэтому, если вы заметили у себя дефекты слуха или хотите действительно проверить свой слух, обратитесь лучше к своему врачу).

Для получения звука в тесте используется функция sound(), которая генерирует непродолжительное звучание специфицированной ноты. Как показано ниже, эта функция содержит все необходимое для того, чтобы сгенерировать любой звук с помощью динамика компьютера.

```
        /* Звучание динамика на заданной частоте */
void sound(freq)
int freq;
{
  unsigned i;
  union {
     long divisor;
     unsigned char c[2];
  } count;
  unsigned char p;
  count.divisor = 1193280 / freq; /* вычисление небходимого
                                     значения счетчика */
  outportb(67,182); /* обращение к таймеру 8253 после
                       установки счетчика */
  outportb(66,count.c[0]); /* пересылка младшего байта */
  outportb(66,count.c[1]); /* пересылка старшего байта */
  p = inportb(97); /* чтение существующего шаблона бит */
  outportb(97,p|3); /* установка битов 0 и 1 */
  for (i=0;i<64000;++i); /* цикл задержки */
  outportb(97,p); /* восстановление первоначального значения
                     шаблона бит для отключения динамика */
}
```

Заметим, что частота звучания ноты специфицирована как аргумент функции. Цикл задержки необходим, так как без него вы бы услышали только мгновенный "щелчок" или "писк". Вы можете изменить частоту работы системного таймера процессора вашего компьютера. При этом, оформив его как параметр функции, вы добьетесь определенной эффективности вашей программы. Функция sound() может использоваться и для получения банального "пищания" компьютера.

Управляющая функция для программы теста слуха представлена ниже.

```
        /* Простейший тест слуха */
#include "dos.h"
void sound();
main()
{
     int freq;
     do {
       printf(" Введите частоту ( 0 - выход ): ");
       scanf("%d",&freq);
       if( freq ) sound(freq);
     } while(freq);
}
```

При использовании теста, в возрастающем порядке указывайте

частоту звука до тех пор, пока звук воспринимается на слух. Для выхода введите 0.

Имитация звука сирены и взврывы.

Вы можете использовать возможность управления динамиком для создания различных звуковых эффектов, которые, в частности, делают видеоигры очень интересными и привлекательными. В основе всех звуковых эффектов лежит варьирование частоты звука - часто самым необычным образом.

Например, для создания эффекта звучания сирены вы должны варьировать частоту звука между двумя конечными точками. Высота звука должна изменяться от меньшей к большей, а затем уменьшаться от большей к меньшей. Функция siren(), представленная ниже, использует этот метод для создания эффекта звучания сирены.

```
#define DELAY 10000
/* Создание эффекта звучания сирены */
void siren()
{
  unsigned i,freq;
  union {
     long divisor;
     unsigned char c[2];
  } count;
  unsigned char p;
  p = inportb(97); /* чтение существующего шаблона бит */
  outportb(97,p|3); /* установка бит 0 и 1 */
     /* повышение звука сирены */
  for (freq = 1000;freq<3000;freq+=RATE) {
     count.divisor = 1193280 / freq; /* вычисление нужного
                                        значения счетчика */
     outportb(67,182); /* обращение к таймеру 8253 после
                          определения значения счетчика */
     outportb(66,count.c[0]); /* пересылка младшего байта */
     outportb(66,count.c[1]); /* пересылка старшего байта */
     for (i=0;i<DELAY;++i);
  }
     /* понижение звука сирены */
  for (;freq>1000;freq-=RATE) {
     count.divisor = 1193280 / freq; /* вычисление нужного
                                        значения счетчика */
     outportb(67,182); /* обращение к таймеру 8253 после
                          определения значения счетчика */
     outportb(66,count.c[0]); /* пересылка младшего байта */
     outportb(66,count.c[1]); /* пересылка старшего байта */
     for (i=0;i<DELAY;++i);
  }

  outportb(97,p); /* восстановление начального вида шаблона
                     бит для отключения динамика */
}
```

Вы можете переопределить значение макроса DELAY в зависимости от производительности вашего компьютера и вашего вкуса. Как вы видите, функция siren() выполняет один полный цикл звучания сирены и на этом завершает свою работу. Для получения эффекта продолжительного звучания сирены вам надо поместить обращение к siren() в цикл.

Для имитации звука взрыва, который используется во многих видеоиграх, можно модифицировать функцию siren() таким образом, чтобы она позволяла генерировать звук лишь нисходящей частоты. Функция laser(), представленная ниже, позволяет получить этот эффект.

```
#define DELAY 10000
/* получение эффекта взрыва */
```

```
void laser()
{
  unsigned i,freq;
  union {
     long divisor;
     unsigned char c[2];
  } count;
  unsigned char p;
  p = inportb(97); /* чтение существующего шаблона бит */
  outportb(97,p|3); /* установка бит 0 и 1 */
     /* взрыв */
     for (;freq>1000;freq-=RATE) {
     count.divisor = 1193280 / freq; /* вычисление нужного
               значения счетчика */
     outportb(67,182); /* обращение к таймеру 8253 после
               определения значения счетчика */
     outportb(66,count.c[0]); /* пересылка младшего байта */
     outportb(66,count.c[1]); /* пересылка старшего байта */
     for (i = 0;i<DELAY;++i);
  }
  outportb(97,p); /* восстановление начального вида шаблона
                     бит для отключения динамика */
}
```

После небольшого экспериментирования вы сможете сами создавать широкий спектр звуковых эффектов. Интерес представляет варьирование скоростью изменения частоты звука для получения определенных эффектов.

Создание "космической музыки".

Соединив воедино произвольное количество стандартных функций Си rend() и sound(), вы создадите "космическую" музыку. Звук, получаемый при выполнении программы, представленной ниже, напоминает "музыку звезд" в старых научно-фантастических фильмах. Несмотря на то, что все звуки генерируются произвольным образом, ритм и рисунок мелодии, возникающие время от времени, действительно оставляют впечатление "небесной музыки".

```
     /* Космическая музыка звезд */
#define DELAY 64000
#include "dos.h"
void sound();
main()
{
     int freq;
     do {
          do {
               freq = rand();
          } while (freq>5000); /* после персонального
                                  прослушивания */
          sound(freq);
     } while (!kbhit());
}
/* звучание динамика на специфицированной частоте */
void sound(freq)
int freq;
{
  unsigned i;
  union {
     long divisor;
     unsigned char c[2];
  } count;
  unsigned char p;
  count.divisor = 1193280 / freq; /* вычисление нужного
                                     значения счетчика */
```

```
        outportb(67,182); /* обращение к таймеру 8253 после
                             определения значения счетчика */
        outportb(66,count.c[0]); /* пересылка младшего байта */
        outportb(66,count.c[1]); /* пересылка старшего байта */
        p = inportb(97); /* чтение существующего шаблона бит */
        outportb(97,p|3); /* установка бит 0 и 1 */

        for (i = 0;i<DELAY;++i); /* задержка 64000 для 10+ Мгц
                                    компьютеров
                                 32000 для 6 МГц PC/AT
                                 20000 для стандарта PC и XT */
        outportb(97,p); /* восстановление начального вида
             шаблона бит для отключения динамика */
    }
```

Эта программа генерирует звуки частотой менее 5000 Гц, так как звуки именно в пределах этой частоты наиболее мягко воспринимаются слухом и не выходят за границы, воспринимаемые ухом человека.

Рекомендуем вам поэкспериментировать с этой программой, установив произвольную длину интервала времени между звуками или фильтруя значения, передаваемые в sound(). Возможны и другие варианты развития вашего творчества.

ГЛАВА 9

ИНТЕРФЕЙС С "МЫШЬЮ"

Наиболее популярным устройством ввода данных после клавиатуры является "мышь" (mouse). Несмотря на то, что "мышь" и сходные технологии, такие как "roller ball", получили широкое распространение лишь в последнее время, популярность "мыши" берет свое начало с момента выхода на рынок очередной разработки фирмы Apple компьютера Apple Lisa, в котором впервые была применена технология "мышь" для работы с пиктограммным (иконным) интерфейсом операционной системы этого компьютера. Модель Apple Lisa произвела форменный переворот в фирме Macintosh, которая пошла по пути использования "мыши" и пиктограммного интерфейса в своих программных продуктах. Перед выходом на рынок серии IBM PS/2 "мышь", по существу, была третьим дополнением к PC. Тем не менее уже при анонсировании системы IBM PS/2 сообщалось, что она снабжена портом для подключения "мыши", и "мышь" занимает значительное место среди PC.

Наилучшее использование "мыши" - предмет постоянных дискуссий. Не все программисты (или пользователи) воспринимают пиктограмный интерфейс. В связи с тем, что "мышь" впервые была использована именно с пиктограммным интерфейсом, вопросы использования "мыши" чаще всего сводятся именно к нему, а это в, свою очередь, отталкивает от "мыши" большой круг программистов, негативно относящихся к такому интерфейсу. Однако "мышь" ведь может использоваться и без такого интерфейса. Например, практически все согласны с тем, что "мышь" эффективно может использоваться при работе с интерактивной графикой.

Некоторые модели манипуляторов типа "мышь", равно как и выполняемые ими функции, могут значительно отличаться друг от друга. Поэтому заметим, что все программы, приведенные в этой главе, ориентированы на использование "мыши" фирмы Microsoft, которая функционально идентична "мыши", используемой в моделях PS /2. Для обеспечения интерфейса с "мышью" фирмы Microsoft вам необходимо иметь по крайней мере саму "мышь", руководство пользователя по программному обеспечению "мыши" (Microsoft Mouse Programmer's Reference Guide) и поставляемый с этим руководством диск. На этом диске расположена специальная библиотека с именем MOUSE.LIB, выполняющая поддержку функционирования "мыши" на самом нижнем уровне. Мы будем использовать функции из этой библиотеки в качестве базовых функций при рассмотрении программ, предлагаемых

вам в этой главе. вы должны помнить, что ваш компилятор С должен быть совместим с подключаемыми на этапе редактирования связей подпрограммами из библиотеки подпрограмм поставляемых на диске фирмой Microsoft. вам также надлежит помнить, что обязательно необходимо наличие драйвера устройства MOUSE.SYS.

После беглого обзора основ использования манипулятора типа "мышь" в этой главе вы получите информацию о том, как можно модифицировать ранее рассмотренную программу "рисования" с тем, чтобы она могла работать с "мышью" (с учетом того, что основные концепции применения "мыши", а также разработанные в процессе

изложения материала подпрограммы, могут использоваться вами в дальнейшем при создании различных конкретных приложений).

НЕКОТОРЫЕ НАЧАЛЬНЫЕ СВЕДЕНИЯ О "МЫШИ"

Для того, чтобы использовать "мышь", прежде всего необходимо инсталировать соответствующий драйвер. Для "мыши" фирмы Microsoft в файл CONFIG.SYS должна быть добавлена следующая строка:

```
device = mouse.sys
```

Для инсталяции драйвера "мыши" фирмы IBM должна быть запущена программа MOUSE.COM. С этой целью в файл AUTOEXEC.BAT может быть добавлена строка вида:

```
mouse
```

После того, как драйвер размещен в системе, любые действия по перемещению "мыши" или нажатию ее клавиш будут вызывать генерацию прерывания 33H. Процесс, управляющий "мышью", вызывает прерывание, затем устанавливает значения соответствующих внутренних переменных и продолжает свою работу. Вследствие того, что прерывание генерируется лишь при изменении "мышью" своего положения, неподвижная "мышь" не вызывает необходимости выделения на нее ресурсов компьтера.

Точно так же, как с клавиатурой ассоциирован курсор, с "мышью" также ассоциирован некий маркер на экране (называемый далее указатель). Подпрограммы в библиотеке в поддержке "мыши" фирмы Microsoft определяют по умолчанию следующую форму указателя: в графических режимах - это стрелка, а в текстовых режимах - прямоугольник в рамках габаритных размеров символа. Аналогично курсору клавиатуры указатель-курсор "мыши" может отображаться лишь тогда, когда "мышь" непосредственно используется. В противном случае указатель-курсор "мыши" не отображается на экране так, как будто интерфейса с "мышью" вообще не существует.

Несмотря на то, что "мышь" и экран физически отделены друг от друга, связь между ними все же имеется, так как драйвер "мыши" автоматически отслеживает содержимое счетчиков, индицирующих текущее состояние указателя-курсора "мыши" на экране. При перемещении "мыши" курсор автоматически перемещается на экране в точном соответствии с изменением координат "мыши". Расстояние, на которое была перемещана "мышь", измеряется в "мышиных" шагах. Один такой шаг равен 1/200 дюйма. Однако в большинстве случаев знать на сколько переместилась "мышь" совсем необязательно.

ВИРТУАЛИЗАЦИЯ И РЕАЛЬНЫЙ ЭКРАН

Библиотека подпрограмм поддержки "мыши" фирмы Microsoft работает с виртуальным экраном в виде массива точек растра (массива из единиц минимального изображения, цвет и яркость которых можно задать независимо от остального изображения), который может отличаться от реального экрана. При перемещении "мыши" счетчики местоположения курсора изменяют свое значение. Перед отображением курсора виртуальные координаты курсора преобразуются в координаты реального экрана. В видеорежимах 6,

14, 15 и 16 это преобразование осуществляется один к одному. В режимах 4 и 5 не каждая точка виртуальной горизонтальной позиции преобразуется в координты реального экрана, а через одну. вы должны обратить внимание на этот факт, так как программа рисования, к которой будет добавлен интерфейс с "мышью" и которая будет рассматриваться в данной главе, работает именно в 4 графическом режиме.

БИБЛИОТЕКА ПОДДЕРЖКИ "МЫШИ"

Подпрограммы внутри MOUSE.LIB ассоциируются у пользователя с одной функцией, использующей в качестве входного аргумента число, специфицирующее номер функции поддержки "мыши". (Этот процесс некоторым образом сходен с процессом доступа к функциям DOS посредством прерывания 21H с указанием номера нужной функции). Имя этой функции определяется моделью памяти, которая используется при компиляции вашей программы. Используйте имя cmouses() для модели маленькой памяти, cmousec() для модели компактной памяти, cmousem() для модели средней памяти и cmousel() для модели большой и самой большой (огромной) памяти. (Заметим, что функция не может работать в модели самой маленькой памяти). Пример, представленный в этой главе, использует модель маленькой памяти, однако вы можете изменить тип модели памяти по своему усмотрению.

Основным форматом функции cmouses() является:

```
void cmouses(fnum, arg2, arg3, arg4);
     int *fnum, *arg2, *arg3, *arg4;
```

Как видно, fnum является номером функции поддержки "мыши", которую необходимо вызвать. Другие параметры содержат информацию, необходимую для спецификации функции. Обратите внимание, что функции передаются не сами аргументы, а указатели на их значения. Функция cmouses() возвращает результаты работы в виде параметров и, следовательно, нуждается в их адресации. Фирма Microsoft определила тридцать функций поддержки "мыши". Однако в программе рисования будут использованы лишь пять из них. Ниже приведен краткий обзор функций поддержки "мыши" фирмы Microsoft, которые будут использованы нами в этой главе.

Привести в исходное состояние, выдать статус.

Функция 0 приводит "мышь" в начальное состояние (сбрасывает "мышь") Она перемещает курсор-указатель "мыши" в центр экрана и "выключает " его. Функция возвращает номер нажатой клавиши "мыши" в качестве значения arg2. После завершения функции fnum принимает значение 0, если "мышь" и соответствующее программное обеспечение не инсталированы, и -1 в противном случае.

Отобразить курсор

Функция 1 отображает указатель-курсор "мыши". Она не возвращает никакого значения.

Переместить курсор

Функция 2 перемещает курсор по экрану. Она не возвращает никакого значения.

Выдать статус клавиши и позицию курсора

Функция 3 возвращает статус клавиши в arg2, виртуальную горизонтальную позицию курсора в arg3, а виртуальную вертикальную позицию курсора в arg4.

Статус клавиши кодируется в битах 0 и 1 байта arg2. Если значение бита 0 установлено (равно 1), то была нажата левая клавиша "мыши", если значение бита 1 установлено (равно 1), то была нажата правая клавиша. Если значения обоих битов не

установлены (равны 0), то никакая клавиша нажата не была.

Установить координаты курсора

Функция 4 устанавливает месторасположение курсора "мыши". Значение arg3 определяет горизонтальную позицию, а значение arg4 - вертикальную позицию курсора. вы всегда должны помнить, что значения не должны выходить за пределы виртуального экрана, который вы используете.

Индикация движения

Функция 11 возвращает число вертикальных и горизонтальных "мышиных" шагов, которое "мышь" прошла со времени последнего обращения к функции 11, другими словами - это изменение вертикальных и горизонтальных координат "мыши". Функция также сбрасывает внутренний регистр-счетчик в 0. Значение вертикального счетчика возвращается в arg3, а горизонтального - в arg4. Это позволяет, если "мышь" после последнего обращения к функции не перемещалась на плоскости, получить значения как горизонтального, так и вертикального счетчиков равными 0. Если значение одного из счетчиков (или обоих) отлично от 0, то "мышь" перемещалась на плоскости.

Положительное значение изменения вертикальных "мышиных" шагов позволяет заключить, что "мышь" двигалась вниз. Отрицательное же значение свидетельствует о движении "мыши" вверх.

Положительное значение изменения числа горизонтальных шагов свидетельствует о том, что "мышь" перемещалась вправо, а отрицательное значение показывает на то, что "мышь" перемещалась влево.

ФУНКЦИИ ПОДДЕРЖКИ "МЫШИ" ВЕРХНЕГО УРОВНЯ

Используя функцию cmouses() вы можете создать набор функций языка Си высокого уровня, которые значительно облегчат вам программирование интерфейсов, ориентированных на использованием "мыши". Посмотрите, как это делается.

Установка "мыши" в исходное состояние.

Функция, представленная ниже, mouse_reset() используется для установки "мыши" в исходное состояние. Заметим, что функция требует наличия соответствующего программного обеспечения и аппаратной части компьютера, а также инсталяции двухклавишной "мыши".

```
/* Установка "мыши" в исходное состояние */
void mouse_reset()
{
     int fnum, arg2, arg3, arg4;
     fnum = 0;  /* Установка "мыши" в исходное состояние  */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     if(fnum!=-1) {
     printf("Аппаратные или программные средства поддержки ");
     printf("'мыши' не инсталированы");
          exit(1);
     }
     if(arg2!=2) {
  printf("Разрешено использование только двухклавишной 'мыши'");
          exit(1);
     }
}
```

Отображение и перемещение курсора "мыши".

Взаимодополняющие друг друга функции cursor_on() и

cursor_off(), представленные ниже, позволяют активизировать и деактивизировать изображение курсора на экране дисплея.

```
/*  Включение курсора "мыши"  */
void cursor_on()
{
     int fnum;
     fnum = 1; /* отобразить курсор */
     cmouses( &fnum,  &fnum,  &fnum,  &fnum);
}

/*  Выключение курсора "мыши"  */
void cursor_off()
{
     int fnum;
     fnum = 2; /* стереть курсор */
     cmouses( &fnum,  &fnum,  &fnum,  &fnum);
}
```

Какая из клавиш "мыши" была нажата?

Другой парой взаимодополняющих друг друга функций являются функции rightb_pressed() и leftb_pressed(), представленные ниже. Эти функции возвращают значение "истина", если нажата правая или левая клавиши.

```
/* Возвращает значение "истина", если нажата правая клавиша,
     и "ложь" в противном случае                              */
rightb_pressed()
{
     int fnum, arg2, arg3, arg4;
     fnum = 3;   /* Чтение позиции и статуса клавиши */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     return arg2 & 2;
}
/* Возвращает значение "истина", если нажата левая клавиша,
     и "ложь" в противном случае                              */
leftb_pressed()
{
     int fnum, arg2, arg3, arg4;
     fnum = 3;   /* Чтение позиции и статуса клавиши */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     return arg2 & 1;
}
```

Как обнаружить перемещение "мыши"?

Функция 11, которая возвращает изменение значения счетчика "мыши" (в "мышиных" шагах) после последнего обращения к ней, позволяет определить факт перемещения "мыши". Функция mouse_motion(), представленная ниже, возвращает изменение местоположения "мыши" в горизонтальном и вертикальном направлениях в переменных, чьи указатели являются аргументами функции. Если оба значения deltax и deltay равны 0, то факт движения "мыши" не регистрируется.

```
/*  Возвращает направление движения  */
void mouse_motion(deltax, deltay)
char *deltax, *deltay;
{
     int fnum, arg2, arg3, arg4;
     fnum = 11; /* получить направление движения */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     if(arg3>0) *deltax = RIGHT;
     else if(arg3<0) *deltax = LEFT;
     else *deltax = NOT_MOVED;
     if(arg4>0) *deltay = DOWN;
     else if(arg4<0) *deltay = UP;
```

```
     else *deltay = NOT_MOVED;
}
```

Макросы RIGHT, LEFT, UP, DOWN и NOT_MOVED определены следующим образом:

```
#define NOT_MOVED 0
#define RIGHT     1
#define LEFT      2
#define UP        3
#define DOWN      4
```

Чтение и установка позиции курсора.

Функции set_mouse_position() и mouse_position(), представленные ниже, используются для установки чтения текущей позиции курсора "мыши".

```
/* Установить координаты курсора "мыши" */
void set_mouse_position(x, y)
int x, y;
{
     int fnum, arg2;
     fnum = 4; /* установка позиции */
     cmouses(&fnum, &arg2, &x, &y);
}
/* Возвращает координаты курсора "мыши" */

void mouse_position(x, y)
int *x, *y;
{
     int fnum, arg2, arg3, arg4;
     fnum = 3; /* получить позицию и статус клавиши */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     *x = arg3;
     *y = arg4;
}
```

ПРОСТЕЙШАЯ ДЕМОНСТРАЦИОННАЯ ПРОГРАММА

Программа, представленная ниже, демонстрирует применение функций поддержки "мыши" высокого уровня. вы можете сразу ввести ее в ваш компьютер и попробовать ее в деле.

```
/*  Интерфейс с "мышью" Microsoft/IBM */
#include "dos.h"
#define NOT_MOVED 0
#define RIGHT     1
#define LEFT      2
#define UP        3
#define DOWN      4
void mouse_position(), mode(), goto_xy(), mouse_motion();
void cursor_on(), cursor_off(), mouse_reset();
main(argc, argv)
int argc;
char *argv[];
{
     char deltax, deltay, x, y;
     if(argc!=2) {
          printf(" Используйте формат: mouser <видеорежим> ");
          exit(1);
     }
     mode(atoi(argv[1]));
     mouse_reset();  /* инициализация "мыши"  */
     cursor_on();    /* "включение" курсора   */
     do {
          goto_xy(0, 0);
          if(leftb_presed()) printf("Левая клавиша");
          if(rightb_pressed()) {
```

```
                printf("Правая клавиша");
                mouse_position(&x, &y);
                printf("%d %d - ", x, y);
           }
           /* Отображение местоположения "мыши" */
           mouse_motion(&deltax, &deltay);
           if(deltax || deltay) {
                printf("Перемещение");
                switch(deltax) {
                     case NOT_MOVED: break;
                     case RIGHT: printf("Вправо");
                          break;
                     case LEFT: printf("Влево");
                          break;
                }

                switch(deltay) {
                     case NOT_MOVED: break;
                     case UP: printf("Вверх");
                          break;
                     case DOWN: printf("Вниз");
                          break;
                }
           }
     /* Цикл выполняется пока обе клавиши нажаты одновременно */
     } while(!(leftb_pressed() && rightb_pressed()));
     mode(3);
}
/* Установка видеорежима  */
void mode(mode_code)
int mode_code;
{
     union REGS r;
     r.h.al = mode_code;
     r.h.ah = 0;
     int86(0x10, &r, &r);
}
/* Пересылка курсора в позицию, специфицированную
     координатами x и y */
void goto_xy(x, y)
int x, y;
{
     union REGS r;
     r.h.ah=2;  /* функция адресации курсора  */
     r.h.dl = y; /* координаты столбца */
     r.h.dh = x; /* координаты строки */
     r.h.bh = 0; /* видеостраница */
     int86(0x10, &r, &r);
}
/*********************************************************/
/* Функции, обеспечивающие интерфейс с "мышью"           */
/*********************************************************/
/*  Включение курсора "мыши"  */
void cursor_on()
{
     int fnum;
     fnum = 1; /* отобразить курсор */
     cmouses( &fnum,  &fnum,  &fnum,  &fnum);
}
/*  Выключение курсора "мыши"  */
void cursor_off()

{
     int fnum;
```

```
     fnum = 2; /* стереть курсор */
     cmouses( &fnum,  &fnum,  &fnum,  &fnum);
}
/* Возвращает значение "истина", если нажата правая клавиша,
     и "ложь" в противном случае                              */
rightb_pressed()
{
     int fnum, arg2, arg3, arg4;
     fnum = 3;   /* Чтение позиции и статуса клавиши   */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     return arg2 & 2;
}
/* Возвращает значение "истина", если нажата левая клавиша,
     и "ложь" в противном случае                              */
leftb_pressed()
{
     int fnum, arg2, arg3, arg4;
     fnum = 3;   /* Чтение позиции и статуса клавиши   */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     return arg2 & 1;
}
/*  Возвращает направление движения  */
void mouse_motion(deltax, deltay)
char *deltax, *deltay;
{
     int fnum, arg2, arg3, arg4;
     fnum = 11; /* получить направление движения */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     if(arg3>0) *deltax = RIGHT;
     else if(arg3<0) *deltax = LEFT;
     else *deltax = NOT_MOVED;
     if(arg4>0) *deltay = DOWN;
     else if(arg4<0) *deltay = UP;
     else *deltay = NOT_MOVED;
}
/* Установить координаты курсора "мыши" */

void set_mouse_position(x, y)
int x, y;
{
     int fnum, arg2;
     fnum = 4; /* установка позиции */
     cmouses(&fnum, &arg2, &x, &y);
}
/* Возвращает координаты курсора "мыши" */
void mouse_position(x, y)
int *x, *y;
{
     int fnum, arg2, arg3, arg4;
     fnum = 3; /* получить позицию и статус клавиши */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     *x = arg3;
     *y = arg4;
}
/* Установка "мыши" в исходное состояние   */
void mouse_reset()
{
     int fnum, arg2, arg3, arg4;
     fnum = 0;  /* Установка "мыши" в исходное состояние  */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     if(fnum!=-1) {
     printf("Аппаратные или программные средства поддержки ");
     printf("'мыши' не инсталированы");
          exit(1);
```

```
     }
     if(arg2!=2) {
     printf("Разрешено использование только двухклавишной ");
     printf("'мыши'");
          exit(1);
     }
}
```

Перед использованием этой программы в командной строке DOS специфицируйте видеорежим, в котором вы желаете работать. Программа будет сообщать вам о направлении перемещения "мыши",а также о нажатых клавишах "мыши". Вдобавок ко всему, нажатие правой клавиши будет вызывать отображение текущей позиции X, У. Нажатие обеих клавиш вызовет завершение программы. Попробуйте выполнить эту программу в различных видеорежиах и обратите внимание на возникающие при этом эффекты.

ВВОД ИНФОРМАЦИИ С ПОМОЩЬЮ "МЫШИ" В ПРОГРАММЕ РИСОВАНИЯ

Итак, вы теперь готовы к тому, чтобы разработать подпрограммы, позволяющие с помощью "мыши" управлять программой рисования. Интерфейс с "мышью" может быть добавлен в существующие подпрограммы управления, что, естественно, будет более предпочтительно, чем разработка новых подпрограмм или модификация существующих.

Такой путь выгоден прежде всего тем, что функциональны возможности клавиш управления курсором сохраняются на все 100 процентов, и пользователь в каждой конкретной ситуации может выбрать наиболее подходящее устройство для ввода данных (клавиатура или "мышь").

Прежде, чем "мышь" будет включена как устройство ввода программу рисования, необходимо разработать две подпрограммы, учитывающие специфику "мыши". Первая подпрограмма - wait_on() позволяет реализовать процесс ожидания отпускания (освобождения) специфицированной клавиши пользователем. Анализ подобного рода имеет весьма большое значение, так как соответствующие прерывания генерируются постоянно, пока клавиша не нажата. (Однако невозможно обеспечить такое мгновенное нажатие на клавишу, в результате которого сформировалось бы лишь одно прерывание). Во многих подпрограммах наоборот важно избежать такой ситуации и поэтому в них каждое нажатие на клавишу генерирует (точнее будет сказать кажется, что генерирует) только одно прерывание за то время, пока клавиша нажата. В соответствии с этим, ваша программа должна обращаться к фунции wait_on(), представленной ниже, непосредственно перед выполнением и после того, когда нажата соответствующая клавиша.

```
/* Возвращает 1, если специфицированная клавиша не нажата */
void wait_on(button)
int button;
{
     if(button== LEFTB)
          while(leftb_pressed());
     else
          while(rightb_pressed());
}
```

Макросы LEFTB и RIGHTB, представленные ниже, используются при обращении к wait_on().

```
#define LEFTB     1
#define RIGHTB    2
```

Второй необходимой вам функцией является mouse_menu(). Эта функция отображает однострочное меню и позволяет пользователю осуществлять выбор из него элементов путем перемещения "мыши" на плоскости (и, соответственно, курсора "мыши" по экрану) и нажатия любой клавиши "мыши". Эта функция может работать только в 4

графическом режиме. Функции передается двумерный массив символов, который содержит элементы меню (которые может выбрать пользователь), значение каждого элемента меню (его код),а также координаты X и У отображения меню на экране. Массив символов определяет максимальную длину каждого элемента меню в 19 символов. Функция возвращает в качестве результата номер выбранного пользователем элемента меню, начиная с 0, или -1, если пользователь не выбрал ни один из элементов меню. Когда функция начинает свою работу, то она вначале вычисляет длину в пикселах (элементах растра) каждого элемента меню, после чего резервирует пространство по начальной и конечной точке растра для каждого элемента меню, одновременно запоминая эту информацию в массиве len. (В четвертом графическом режиме каждый символ имеет высоту в 8 точек растра и ширину в 16 точек растра.) После этих вычислений, функция переходит в состояние ожидаения прерывания от клавиш "мыши". При этом осуществляется анализ нажата или нет клавиша "мыши" в момент нахождения ее курсора в области меню, и, если да, то в месте расположения какого элемента меню. Функция mouse_menu() приведена ниже.

```
/* Отображает однострочное меню для "мыши" и возвращает
   код выбранного пользователем элемента меню          */
mouse_menu(count, item, x, y)
int count;           /* количество элементов меню */
char item[][20];      /* элементы меню */
int x, y;           /* позиции отображения */
{
     int i, len[MENU_MAX][2], t;
     int mousex, mousey;
     goto_xy(x, y);
     t = 0;
     for(i=0; i<count; i++) {
          printf("%s   ", item[i]);
          len[i][0] = t;
          /* каждый символ имеет ширину в 16 точек растра */
          len[i][1] = t + strlen(item[i])*16;
          t = len[i][1] + 32; /* добавляется два пробела между
                                 элементами меню */
     }
     /* ожидание выбора пользователем элемента меню */
     do {
          if(rightb_pressed() || leftb_pressed()) break;
     } while(!kbhit());
     /* ожидание нажатия клавиши */
     while(rightb_pressed() || leftb_pressed());
     /* получить текущую позицию курсора "мыши" */
     mouse_position(&mousex, &mousey);
     /* анализируется, находится ли курсор в пределах меню */

          if(mousey>=0 && mousey<8)  /* символ имеет высоту
                                 8 точек растра */
               for(i=0; i<count; i++) {
                    if(mousex>len[i][0] && mousex<len[i][1])
                         return i;
               }
          return i;
     }
      returtn -1; /* выбор из меню не осуществлялся */
  }
```

ОСНОВНОЙ ЦИКЛ РАБОТЫ ПРОГРАММЫ

В отличие от всей остальной части программы рисования, фрагмент функции main() претерпел наиболее зеначительные

изменения. В функцию main() добавлены, в частности, функции, обеспечивающие интерфейвс с мышью. Текст функции main приведен ниже.

```
main()
{
     char done=0;
     mode(4); /* переключатель в 4 графический режим для
               адаптеров CGA/EGA  */
     palette(0); /* палитра 0  */
     mouse_reset(); /* инсталяция "мыши" */
     xhairs(x, y); /* установить графический курсор */
     set_mouse_position(y*2, x); /* установить начальную позицию
                                    курсора "мыши" */
     do {
          /* просматривается, перемещалась ли "мышь" */
          mouse_motion(&deltax, &deltay);
          if(deltax || deltay) read_mouse();
          /* проверка нажатия клавиши */
          if(leftb_pressed() || rightb_pressed())
               read_mouse();
          if(kbhit()) {
               done = read_kb();
          /* перемещение "мыши" в соответствием с размещением
               графического курсора */
               set_mouse_position(y*2, x);
          }
     } while (!done);
     mode(2);
}
```

Как вы можете видеть, функции main(), за исключением функции done(), не оперируют локальными переменными. Вместо этого все управляющие и вычисляемые переменные, необходимые для работы программы, являются глобальными, что позволяет как функциям работы с клавиатурой, так и функциям работы с "мышью", избавится от длинного списка аргументов при обращении к ним, а также упростить их использование в программе. Как вы видите, анализ типа прерывания (от клавиатуры или от "мыши") осуществляется внутри цикла. При поступлении прерывания от этих устройств вызываются соответствующие функции. Заметьте, что курсор "мыши" при этом не выключается. Взамен этого во время выполнения рисующей части программы графический курсор используется для индикации текущей позиции на экране. Курсор "мыши" активизируется лишь при работе с однострочным меню.

Функция read_kb(), которая обеспечивает процесс ввода с клавиатуры, представлена ниже. В основном она ничем не отличается от оригинальной версии и выполняет довольно значительные по своей роли функции.

```
/* Чтение и обработка команд, вводимых с клавиатуры */
read_kb()
{
     union k{
          char c[2];
          int i;
     } key;
     key.i = bioskey(0);
     xhairs(x, y); /* стереть графический курсор */
     if(!key.c[0]) switch(key.c[1]){
          case 75:  /* влево */
               if(on_flag) line(x, y, x, y-inc, cc);
               y -= inc;
               break;
          case 77:  /* вправо */
               if(on_flag) line(x, y, x, y+inc, cc);
```

```
            y += inc;
            break;
        case 72:  /* вверх */
            if(on_flag) line(x, y, x-inc, y, cc);
            x -= inc;
            break;
        case 80:  /* вниз */
            if(on_flag) line(x, y, x+inc, y, cc);
            x += inc;
            break;
        case 71:  /* влево вверх */
            if(on_flag) line(x, y, x-inc, y-inc, cc);
            y -= inc; x -= inc;
            break;
        case 73:  /* вправо вверх */
            if(on_flag) line(x, y, x-inc, y+inc, cc);
            y += inc; x -= inc;
            break;
        case 79: /* влево вниз */
            if(on_flag) line(x, y, x+inc, y-inc, cc);
            y -= inc; x += inc;
            break;
        case 81:  /* вправо вниз */
            if(on_flag) line(x, y, x+inc, y+inc, cc);
            y += inc; x += inc;
            break;
        case 59: inc = 1; /* F1 - уменьшить скорость */
            break;
        case 60: inc = 5; /* F2 - увеличить скорость */
            break;

    }
    else switch(tolower(key.c[0])) {
        case 'o': on_flag = !on_flag; /* переключателоль
                            шаблона цвета */
            break;
        case '1': cc = 1; /* цвет 1 */
            break;
        case '2': cc = 2; /* цвет 2 */
            break;
        case '3': cc = 3; /* цвет 3 */
            break;
        case '0': cc = 0; /* цвет 0 */
            break;
        case 'b': box(startx, starty, endx, endy, cc);
            break;
        case 'f': fill_box(startx, starty, endx, endy, cc);
            break;
        case 'l': line(startx, starty, endx, endy, cc);
            break;
        case 'c': circle(startx, starty, endy-starty, cc);
            break;
        case 'h': fill_circle(startx, starty, endy-starty, cc);
            break;
        case 's': save_pic();
            break;
        case 'r': load_pic();
            break;
        case 'm': /* переместить область */
            move(startx, starty, endx, endy, x, y);
            break;
        case 'x': /* копировать область */
            copy(startx, starty, endx, endy, x, y);
            break;
```

```
          case 'd': /* определить объект для вращения */
               sides = define_object(object, x, y);
               break;
          case 'a': /* вращать объект */
               rotate_object(object, 0.05, x, y, sides);
               break;
          case '\r': /* установить конечную точку для линии,
                        окружности или прямоугольника */
               if(first_point) {
                    startx = x; starty = y;
               }
               else {
                    endx = x; endy = y;
               }
               first_point = !first_point;
               break;
          case 'p': pal_num = pal_num==1 ? 2:1;
               palette(pal_num);
     }
     /* восстановить изображение графического курсора */
     xhairs(x, y);

     if(tolowel(key.c[0])=='q') return 1;
     return 0;
}
```

Функция read_mouse(), которая обрабатывает ввод от "мыши", представлена ниже.

```
/* Чтение и обработка команд, вводимых с помощью "мыши" */
read_mouse()
{
     int oldx, oldy;
     int choice;
     oldx = x; oldy = y;
     xhairs(x, y); /* стереть с текущей позиции */
     /* нажаты обе клавиши для активизации меню */
     if(rightb_pressed() && leftb_pressed()) {
          choice = menu(); /* получить результат выбора
                              из меню */
          switch(choice) {
           case 0: box(startx, starty, endx, endy, cc);
                break;
           case 1: circle(startx, starty, endy-starty, cc);
                break;
           case 2: line(startx, starty, endx, endy, cc);
                break;
           case 3: fill_box(startx, starty, endx, endy, cc);
                break;
           case 4: fill_circle(startx, starty, endy-starty, cc);
                break;
          }
     }
     /* правая клавиша определяет конечную точку фигуры */
     else if(rightb_pressed()) {
          if(first_point) {
               startx = x; starty = y;
          }
          else {
               endx = x; endy = y;
          }
          first_point = !first_point;
          wait_on(RIGHTB); /* ожидание освобождения клавиши */
     }
     if(deltax || deltay) {
          mouse_position(&y, &x);
```

```
          y = y / 2; /* нормализация координат виртуального
                        экрана */
          /* нажмите левую клавишу для рисования */

          if(leftb_pressed()) mouse_on_flag = 1;
          else mouse_on_flag = 0;
          if(mouse_on_flag) line(oldx, oldy, x, y, cc);
     }
     /* восстановить изображение графического курсора */
     xhairs(x, y);
}
```

Функция read_mouse() работает следующим образом. Во-первых, анализируется факт одновременного нажатия двух клавиш. Если факт имеет место, то путем обращения к функции menu() активизируется меню, которое устанавливается и управляется путем обращения к mouse_menu(). Если пользователь осуществил выбор одного из элементов меню, то инициируется соответствующее действие. Так как в состав текущего меню входят различные фигуры, из которых складывается изображение, - прямоугольники, окружности, линии и действия, позволяющие, например, закрашивать эти фигуры, то с помощью "мыши" вы можете выбрать один из этих элементов будущего рисунка или действие. (вы можете также включить в меню средства, позволяющие пользователю выбирать цвета и устанавливать палитру).

Правая клавиша используется для определения конечной точки линий, прямоугольников и окружностей, точно так же, как клавиша <ВВОД> используется для этих же целей при работе с клавиатурой. Установите "мышь" в первой конечной точке объекта и нажмите правую клавишу. Затем переместите "мышь" во вторую конечную точку и опять нажмите клавишу. Так вы получите изображение интересующей вас фигуры.

Когда левая клавиша не нажата, "мышь" может перемещаться по экрану, не оставляя за собой никакого следа. Это происходит потому, что по умолчанию перо "мыши" считается поднятым. Для того, чтобы получить возможность "писать мышью" (опустить перо), необходимо нажать левую клавишу. Пока клавиша нажата, перо считается опущенным.

В конце концов, если "мышь" изменила свои координаты, будут обновлены значения счетчиков X и Y.

Функция menu(), представленная ниже, использует для установки и управления меню функцию mouse_menu(), описанную ранее.

```
/*  Отображение меню */
menu()
{
     register int i, j;
     char far *ptr = (char far *) 0xB8000000; /* Указатель на
                                                 CGA-память */
     char far *temp;
     unsigned char buf[14][80]; /* для хранения содержимого
                                   экрана */
     int x, y, choice;
     char items[][20] = {

          "BOX",
          "CIRCLE",
          "LINE",
          "FILL BOX",
          "FILL CIRCLE"
     };
     temp = ptr;
     /* сохранение верхней части текущего экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
               buf[i][j] = *temp; /* четный байт */
```

```
                buf[i][j+1] = *(temp+8152); /* нечетный байт */
                *temp = 0; *(temp+8152) = 0; /* очистка верхней
                                                части экрана */
                temp++;
          }
     goto_xy(0, 0);
     /* ожидание, которое будет прервано в результате нажатия на
        клавишу */
     while(rightb_pressed() || leftb_pressed());
     cursor_on();
     choice = mouse_menu(5, items, 0, 0);
     cursor_off();
     temp = ptr;
     /* восстановление изображения верхней части экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
                *temp = buf[i][j];
                *(temp+8152) = buf[i][j+1];
                temp++;
          }
     return choice;
}
```

В процессе работы функция прежде всего сохраняет изображение верхней части текущего экрана, не ожидая нажатия какой-либо клавиши. Затем следует "включение" курсора "мыши" и обращение к функции mouse_menu(). После передачи управления от функции mouse_menu() курсор включается, восстанавливается предыдущее изображение верхней части экрана, и функция возвращает соответствующий код выбранного элемента меню.

ОПРЕДЕЛЕНИЕ ОБЪЕКТОВ С ПОМОЩЬЮ "МЫШИ"

Оригинальная версия программы рисования допускает использование объектов, определяемых с помощью функции define_object(), путем специфицирования конечных точек каждого сегмента линии. Каждый определенный ранее объект может быть подвергнут вращению. Версия функции define_object(), представленная ниже, также позволяет с помощью "мыши" указывать конечные точки (габариты) объектов.

```
/* Определение объекта по его габаритным точкам с помощью
   клавиатуры или "мыши" */
define_object(ob, x, y)
double ob[][4];
int x, y;
{
     union k{
          char c[2];
          int i;
     } key;
     register int i, j;
     char far *ptr = (char far *) 0xB8000000; /* Указатель на
                              CGA-память */
     char far *temp;
     unsigned char buf[14][80];
     int sides=0;
     int deltax, deltay, oldx, oldy;
     temp = ptr;
     /* сохранение верхней части текущего экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
                buf[i][j] = *temp; /* четный байт */
                buf[i][j+1] = *(temp+8152); /* нечетный байт */
                *temp = 0; *(temp+8152) = 0; /* очистка верхней
                                                части экрана */
```

```
                temp++;
            }
        i = 0;
        key.i = 0;
        xhairs(x, y);
        do {
            goto_xy(0, 0);
            printf("Определите сторону %d", sides+1);
            if(i==0) printf("Укажите первую габаритную точку");
            else printf("Укажите вторую габаритную точку");
            do {
/************** Добавочная часть для мыши. ******************/
            /* Просматривается, если "мышь" перемещается */

                mouse_motion(&deltax, &deltay);
            /* Используйте левую клавишу для определения точки*/
                if(leftb_pressed()) {
                    /* стирание графического курсора */
                    xhairs(x, y);
                    /* запоминание координат точки */
                    ob[sides][i++] = (double) x;
                    ob[sides][i++] = (double) y;
                    if(i==4) {
                        i = 0;
                        sides++;
                    }
                    break;
                }
            } while(!kbhit() && !deltax && ! deltay);
            if(leftb_pressed()) wait_on(LEFTB);
            if(deltax || deltay) {
            /* если "мышь" переместилась, то пересчет координат*/
                oldx = x; oldy = y;
                mouse_position(&y, &x);
                y = y / 2; /* нормализация коодинат виртуального
                              экрана*/
                /* стирание графического курсора */
                xhairs(oldx, oldy);
            }
/********** Конец добавочной части для "мыши" ****************/
            else if(kbhit()) {
                key.i = bioskey(0);
                /* изображение графического курсора */
                xhairs(x, y);
                if(key.c[0]==13) {
                    /* используйте <ВВОД> для определения точки*/
                    ob[sides][i++] = (double) x;
                    ob[sides][i++] = (double) y;
                    if(i==4) {
                        i = 0;
                        sides++;
                    }
                }
               /* если клавиши управления курсором, то перемещение
                    графического курсора */
                if(!key.c[0]) switch(key.c[1]) {
                    case 75: /* влево */
                        y-=1;
                        break;
                    case 77: /* вправо */
                        y+=1;
                        break;
                    case 72: /* вверх */
                        x-=1;
```

```
                    break;

               case 80: /* вниз */
                    x+=1;
                    break;
               case 71: /* влево вверх */
                    x-=1; y-=1;
                    break;
               case 73: /* вправо вверх */
                    x-=1; y+=1;
                    break;
               case 79: /* влево вниз */
                    x+=1; y-=1;
                    break;
               case 81: /* вправо вниз */
                    x+=1; y+=1;
                    break;
          }
     }
     if(key.c[1] != 59) xhairs(x, y);
} while(key.c[1] != 59); /* нажмите F1 для остановки */
temp = ptr;
/* востановление изображения в верхней части экрана */
for(i=0; i<14; i++)
     for(j=0; j<80; j+=2) {
          *temp = buf[i][j];
          *(temp+8152) = buf[i][j+1];
          temp++;
     }
return sides;
}
```

Как вы можете видеть, левая клавиша используется для указания габаритных точек. Движение "мыши" кодируется точно так же, как и в функции read_mouse(). За исключением того, что в функцию добавлен фрагмент исходного текста для поддержки ввода информации с помощью "мыши", оригинальная версия функции define_object() больше никаких изменений не претерпела. Для указания очередной габаритной точки переместите "мышь" в нужную позицию и нажмите клавишу.

ПОЛНЫЙ ТЕКСТ МОДИФИЦИРОВАННОЙ ПРОГРАММЫ РИСОВАНИЯ

Ниже приведен исходный текст программы рисования с учетом всех изменений и дополнений, обсужденных выше.

```
/* Эта версия программы рисования для адаптеров  CGA/EGA
   позволяет использовать "мышь" фирмы Microsoft/IBM,
   а также альтернативное устройство ввода (клавиатуру)
*/
#define NUM_SIDES 20 /* Максимальное число сторон фигуры.
                        По желанию можно увеличить */
#define NOT_MOVED 0
#define RIGHT     1
#define LEFT      2
#define UP        3
#define DOWN      4
#define LEFTB     1
#define RIGHTB    2
#define MENU_MAX 20 /* Максимальное число элементов меню */
#include "dos.h"
#include "stdio.h"
#include "math.h"
void mode(), line(), box(), fill_box();
void mempoint(), palette(), xhairs();
void circle(), plot_circle(), fill_circle();
```

```
void rotate_point(), rotate_object(), goto_xy();
void display_object(), copy(), move();
void save_pic(), load_pic();
void set_mouse_position(), mouse_position(), mouse_motion();
void cursor_on(), cursor_off(), wait_on(), mouse_reset();
unsigned char read_point();
/* Это массив предназначен для хранения координат динамически
      определяемой фигуры   */
double object[NUM_SIDES][4];
double asp_ratio; /* коэффициент сжатия окружностей */
     int x=10, y=10; /* текущая позиция экрана */
     int cc=2; /* текущий цвет */
     int on_flag=1, mouse_on_flag=0; /* перо опущено или
                                        поднято */
     int pal_num=1;                  /* номер палитры */
     /* Габаритные точки, определяющие линию, окружность
     или прямоугольник */
     int startx=0, starty=0, endx=0, endy=0, first_point=1;
     int inc=1;       /* приращение движения */

     int sides=0;     /* номер стороны определяемой фигуры */
     int deltax, deltay;    /* "мышь" изменяет данные в
                               индицированной позиции */
main()
{
     char done=0;
     mode(4); /* переключатель в 4 графический режим для
                 адаптеров CGA/EGA  */
     palette(0);    /* палитра 0  */
     mouse_reset(); /* инсталяция "мыши" */
     xhairs(x, y);  /* установить графический курсор */
     set_mouse_position(y*2, x); /* установить начальную позицию
                                    курсора "мыши" */
     do {
          /* просматривается, перемещалась ли "мышь" */
          mouse_motion(&deltax, &deltay);
          if(deltax || deltay) read_mouse();
          /* проверка нажатия клавиши */
          if(leftb_pressed() || rightb_pressed())
               read_mouse();
          if(kbhit()) {
               done = read_kb();
          /* перемещение "мыши" в соответствии с размещением
               графического курсора */
               set_mouse_position(y*2, x);
          }
     } while (!done);
     mode(2);
}
/* Чтение и обработка команд, вводимых с помощью "мыши" */
read_mouse()
{
     int oldx, oldy;
     int choice;
     oldx = x; oldy = y;
     xhairs(x, y); /* стереть с текущей позиции */
     /* нажаты обе клавиши для активизации меню */
     if(rightb_pressed() && leftb_pressed()) {
          choice = menu(); /* получить результат выбора
                              из меню */
          switch(choice) {
            case 0: box(startx, starty, endx, endy, cc);
                 break;
            case 1: circle(startx, starty, endy-starty, cc);
```

```
                    break;
               case 2: line(startx, starty, endx, endy, cc);

                    break;
               case 3: fill_box(startx, starty, endx, endy, cc);
                    break;
               case 4: fill_circle(startx, starty, endy-starty, cc);
                    break;
          }
     }
     /* правая клавиша определяет конечную точку фигуры */
     else if(rightb_pressed()) {
          if(first_point) {
               startx = x; starty = y;
          }
          else {
               endx = x; endy = y;
          }
          first_point = !first_point;
          wait_on(RIGHTB); /* ожидание освобождения клавиши */
     }
     if(deltax || deltay) {
          mouse_position(&y, &x);
          y = y / 2; /* нормализация координат виртуального
                        экрана */
          /* нажмите левую клавишу для рисования */
          if(leftb_pressed()) mouse_on_flag = 1;
          else mouse_on_flag = 0;
          if(mouse_on_flag) line(oldx, oldy, x, y, cc);
     }
     /* восстановить изображение графического курсора */
     xhairs(x, y);
}
/* Чтение и обработка команд, вводимых с клавиатуры */
read_kb()
{
     union k{
          char c[2];
          int i;
     } key;
     key.i = bioskey(0);
     xhairs(x, y); /* стереть графический курсор */
     if(!key.c[0]) switch(key.c[1]){
          case 75:  /* влево */
               if(on_flag) line(x, y, x, y-inc, cc);
               y -= inc;
               break;
          case 77:  /* вправо */
               if(on_flag) line(x, y, x, y+inc, cc);
               y += inc;
               break;
          case 72:  /* вверх */
               if(on_flag) line(x, y, x-inc, y, cc);

               x -= inc;
               break;
          case 80:  /* вниз */
               if(on_flag) line(x, y, x+inc, y, cc);
               x += inc;
               break;
          case 71:  /* влево вверх */
               if(on_flag) line(x, y, x-inc, y-inc, cc);
               y -= inc; x -= inc;
               break;
```

```
        case 73:  /* вправо вверх */
             if(on_flag) line(x, y, x-inc, y+inc, cc);
             y += inc; x -= inc;
             break;
        case 79: /* влево вниз */
             if(on_flag) line(x, y, x+inc, y-inc, cc);
             y -= inc; x += inc;
             break;
        case 81:  /* вправо вниз */
             if(on_flag) line(x, y, x+inc, y+inc, cc);
             y += inc; x += inc;
             break;
        case 59: inc = 1; /* F1 - уменьшить скорость */
             break;
        case 60: inc = 5; /* F2 - увеличить скорость */
             break;
   }
   else switch(tolower(key.c[0])) {
        case 'o': on_flag = !on_flag; /* переключатель
                                         шаблона цвета */
             break;
        case '1': cc = 1; /* цвет 1 */
             break;
        case '2': cc = 2; /* цвет 2 */
             break;
        case '3': cc = 3; /* цвет 3 */
             break;
        case '0': cc = 0; /* цвет 0 */
             break;
        case 'b': box(startx, starty, endx, endy, cc);
             break;
        case 'f': fill_box(startx, starty, endx, endy, cc);
             break;
        case 'l': line(startx, starty, endx, endy, cc);
             break;
        case 'c': circle(startx, starty, endy-starty, cc);
             break;
        case 'h': fill_circle(startx, starty, endy-starty, cc);
             break;
        case 's': save_pic();
             break;
        case 'r': load_pic();
             break;
        case 'm': /* переместить область */

             move(startx, starty, endx, endy, x, y);
             break;
        case 'x': /* копировать область */
             copy(startx, starty, endx, endy, x, y);
             break;
        case 'd': /* определить объект для вращения */
             sides = define_object(object, x, y);
             break;
        case 'a': /* вращать объект */
             rotate_object(object, 0.05, x, y, sides);
             break;
        case '\r': /* установить конечную точку для линии,
                      окружности или прямоугольника */
             if(first_point) {
                  startx = x; starty = y;
             }
             else {
                  endx = x; endy = y;
             }
```

```
                first_point = !first_point;
                break;
           case 'p': pal_num = pal_num==1 ? 2:1;
                palette(pal_num);
     }
     /* восстановить изображение графического курсора */
     xhairs(x, y);
     if(tolowel(key.c[0])=='q') return 1;
     return 0;
}
/* Установить палитру */
void palette(pnum)
int pnum;
{
     union REGS r;
     r.h.bh = 1; /* кодирование 4 графического режима */
     r.h.bl = pnum;
     r.h.ah = 11; /* функцкия установления палитры */
     int86(0x10, &r, &r);
}
/* Установить видеорежим */
void mode(mode_code)
int mode_code;
{
     union REGS r;
     r.h.al = mode_code;
     r.h.ah = 0;
     int86(0x10, &r, &r);
}
/* Рисование прямоугольника */

void box(startx, starty, endx, endy, color_code)
int startx, starty, endx, endy, color_code;
{
     line(startx, starty, endx, starty, color_code);
     line(startx, starty, startx, endy, color_code);
     line(startx, endy, endx, endy, color_code);
     line(endx, starty, endx, endy, color_code);
}
/* Рисование линии в специфицированном цвете с использованием
   основного алгоритма Брезенхама  */
void line(startx, starty, endx, endy, color)
int startx, starty, endx, endy, color;
{
     register int t, distance;
     int x=0, y=0, delta_x, delta_y;
     int incx, incy;
     /* определение расстояния в обоих направлениях */
     delta_x = endx-startx;
     delta_y = endy-starty;
     /* Вычисление направления приращения. Приращение
     вычисляется относительно 0 как для горизонтальной,
     так и для вертикальной линии
     */
     if(delta_x>0) incx = 1;
     else if(delta_x==0) incx = 0;
     else incx=-1;
     if(delta_y>0) incy = 1;
     else if(delta_y==0) incy = 0;
     else incy=-1;
     /* Определяется, какое расстояние больше */
     delta_x = abs(delta_x);
     delta_y = abs(delta_y);
     if(delta_x>delta_y) distance = delta_x;
```

```
     else distance = delta_y;
     /* Вычерчиывние линии */
     for(t=0; t<=distance+1; t++) {
          mempoint(startx, starty, color);
          x+=delta_x;
          y+=delta_y;
          if(x>distance) {
               x-=distance;
               startx+=incx;
          }
          if(y>distance) {
               y-=distance;
               starty+=incy;
          }

     }
}
/* Закрашивание прямоугольника специфицированным цветом */
void fill_box(startx, starty, endx, endy, color_code)
int startx, starty, endx, endy, color_code;
{
     register int i, begin, end;
     begin = startx<endx ? startx : endx;
     end = startx>endx ? startx : endx;
     for(i=begin; i<=end; i++)
          line(i, starty, i, endy, color_code);
}
/* Рисование окружности с использованием целочисленного алгоритма
   Брезенхама */
void circle(x_center, y_center, radius, color_code)
int x_center, y_center, radius, color_code;
{
     register int x, y, delta;
     asp_ratio = 1.0; /* Если Вас не удовлетворяют предлагаемые
                         пропорции фигуры, измените значение этой
                         переменной */
     y = radius;
     delta = 3 - 2 * radius;
     for(x=0; x<y; ) {
          plot_circle(x, y, x_center, y_center, color_code);

          if(delta < 0)
               delta += 4*x+6;
          else {
               delta += 4*(x-y)+10;
               y--;
          }
          x++;
     }
     x=y;
     if(y) plot_circle(x, y, x_center, y_center, color_code);
}
/* Непосредственное вычерчивание окружности по определенным
   пользователем точкам */
void plot_circle(x, y, x_center, y_center, color_code)
int x, y, x_center, y_center, color_code;
{
     int startx, endx, x1, starty, endy, y1;
     starty = y*asp_ratio;
     endy = (y+1)*asp_ratio;

     startx = x*asp_ratio;
     endx = (x+1)*asp_ratio;
     for(x1=startx; x1<endx; ++x1) {
```

```
            mempoint(x1+x_center, y+y_center, color_code);
            mempoint(x1+x_center, y_center-y, color_code);
            mempoint(x_center-x1, y_center-y, color_code);
            mempoint(x_center-x1, y+y_center, color_code);
      }
      for(y1=starty; y1<endy; ++y1) {
            mempoint(y1+x_center, x+y_center, color_code);
            mempoint(y1+x_center, y_center-x, color_code);
            mempoint(x_center-y1, y_center-x, color_code);
            mempoint(x_center-y1, x+y_center, color_code);
      }
}
/* Закрашивание окружности путем циклического обращения к функции
   circle() с меньшим радиусом */
void fill_circle(x, y, r, c)
int x, y, r, c;
{
      while(r) {
            circle(x, y, r, c);
            r--;
      }
}
/* Отображение графического курсора  */
void xhairs(x, y)
int x, y;
{
      line(x-4, y, x+3, y, 1 | 128);
      line(x, y+4, x, y-3, 1 | 128);
}
/* Запись точки непосредственно в CGA/EGA */
void mempoint(x, y, color_code)
int x, y, color_code;
{
      union mask {
            char c[2];
            int i;
      } bit_mask;
      int i, index, bit_position;
      unsigned char t;
      char xor; /* xor - цвет изображения записывается или
                   перезаписывается */
      char far *ptr = (char far *) 0xB8000000; /* Указатель
                                                  CGA памяти */


      bit_mask.i=0xFF3F;  /* 11111111 00111111 в двоичном
                             коде   */
      /* проверка значений для 4 режима */
      if(x<0 || x>199 || y<0 || y>319) return;
      xor= color_code & 128; /* просматривается, если режим
                     установлен */
      color_code = color_code & 127; /* маска старшего бита */
      /* установка маски бита и кода цвета для правой ячейки
         памяти */
       bit_position = y%4;
       color_code<<=2*(3-bit_position);
       bit_mask.i>>=2*bit_position;
       /* поиск байта для корректировки в видеопамяти */
       index = x*40 +(y >> 2);
       if(x % 2) index += 8152; /* если дополнительно
                                   использовался 2-й сегмент
                                   памяти */
       /* запись цвета */
       if(!xor) { /* режим перезаписи */
```

```
            t = *(ptr+index) & bit_mask.c[0];
            *(ptr+index) = t | color_code;
        }
        else { /* режим записи цвета */
            t = *(ptr+index) | (char)0;
            *(ptr+index) = t ^ color_code;
        }
}
/* Непосредственное чтение байта из видеопамяти CGA/EGA
   в режиме 4 */
unsigned char read_point(x, y)
int x, y;
{
     union mask {
          char c[2];
          int i;
     } bit_mask;
     int i, index, bit_position;
     unsigned char t;
     char xor; /* xor - цвет изображения записывается или
                  перезаписывается */
     char far *ptr = (char far *) 0xB8000000; /* Указатель
                                                 CGA памяти */
     bit_mask.i=0xFF3F;  /* 11111111 00111111 в двоичном коде */
     /* проверка значений для 4 режима */
     if(x<0 || x>199 || y<0 || y>319) return;


     /* установка маски бита и кода цвета для правой ячейки
        памяти */
      bit_position = y%4;
      bit_mask.i<<=2*(3-bit_position);
      /* поиск нужного байта в видеопамяти */
      index = x*40 +(y >> 2);
      if(x % 2) index += 8152; /* если дополнительно
                                  использовался 2-й блок
                                  памяти */
     /* чтение цвета */
      t = *(ptr+index) & bit_mask.c[0];
     t >>=2*(3-bit_position);
     return t;
}
/* сохранение видео изображения на экране */
void save_pic()
{
     char fname[80];
     FILE *fp;
     register int i, j;
     char far *ptr = (char far *) 0xB8000000; /* Указатель
                                                 CGA памяти */
     char far *temp;
     unsigned char buf[14][80]; /* буфер содержимого экрана */
     temp = ptr;
     /* сохранение верхней части текущего экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
               buf[i][j] = *temp; /* четный байт */
               buf[i][j+1] = *(temp+8152); /* нечетный байт */
               *temp = 0; *(temp+8152) = 0; /* очистка верхней
                                               части экрана */
               temp++;
          }
     goto_xy(0, 0);
     printf("Имя файла : ");
```

```
     gets(fname);
     if(!(fp=fopen(fname, "wb"))) {
          printf("Файл не может быть открыт\n");
          return;
     }
     temp = ptr;
     /* восстановление изображения верхней части экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {


               *temp = buf[i][j];
               *(temp+8152) = buf[i][j+1];
               temp++;
          }
     /* запись изображения в файл */
     for(i=0; i<8152; i++) {
          putc(*ptr, fp); /* четный байт */
          putc(*(ptr+8152), fp); /* нечетный байт */
          ptr++;
     }
     fclose(fp);
}
/* загрузка в память изображения из файла */
void load_pic()
{
     char fname[80];
     FILE *fp;
     register int i, j;
     char far *ptr = (char far *) 0xB8000000; /* Указатель
                              CGA памяти */
     char far *temp;
     unsigned char buf[14][80]; /* буфер содержимого экрана */
     temp = ptr;
     /* сохранение верхней части текущего экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
               buf[i][j] = *temp; /* четный байт */
               buf[i][j+1] = *(temp+8152); /* нечетный байт */
               *temp = 0; *(temp+8152) = 0; /* очистка верхней
                                               части экрана */
               temp++;
          }
     goto_xy(0, 0);
     printf("Имя файла : ");
     gets(fname);
     if(!(fp=fopen(fname, "rb"))) {
          goto_xy(0, 0);
          printf("Файл не может быть открыт\n");
          temp = ptr;
          /* восстановление изображения верхней части экрана */
          for(i=0; i<14; i++)
               for(j=0; j<80; j+=2) {
                    *temp = buf[i][j];
                    *(temp+8152) = buf[i][j+1];
                    temp++;
               }
          return;
     }


     /* Загрузка изображения из файла */
     for(i=0; i<8152; i++) {
          *ptr = getc(fp); /* четный байт */
```

```
          *(ptr+8152) = getc(fp); /* нечетный байт */
          ptr++;
     }
     fclose(fp);
}
/* Перемещение курсора в специфицированную позицию */
void goto_xy(x, y)
int x, y;
{
     union REGS r;
     r.h.ah=2; /* функция адресации курсора */
     r.h.dl = y; /* координаты столбца */
     r.h.dh = x; /* координаты строки */
     r.h.bh = 0; /* видеостраница */
     int86(0x10, &r, &r);
}
/* Перемещение области в другое место */
void move(startx, starty, endx, endy, x, y)
int startx, starty; /* верхняя левая координата */
int endx, endy; /* нижняя правая координата места, куда будет
                   перемещена область */
int x, y; /* верхняя левая координата области, откуда будет
             перемещено изображение*/
{
     int i, j;
     unsigned char c;
     for(; startx<=endx; startx++, endx++)
          for(i=starty, j=y; i<=endy; i++, j++) {
               c = read_point(startx, i); /* чтение точки */
               mempoint(startx, i, 0); /* стирание предыдущего
                                          изображения */
               mempoint(x, j, c); /* запись его в новое место */
          }
}
/* Копировать область в другое место */
void copy (startx, starty, endx, endy, x, y)
int startx, starty; /* верхняя левая координата */
int endx, endy; /* нижняя правая координата места, куда будет
                   скопирована область */
int x, y; /* верхняя левая координата области, откуда будет


             перемещено изображение */
{
     int i, j;
     unsigned char c;
     for(; startx<=endx; startx++, endx++)
          for(i=starty, j=y; i<=endy; i++, j++) {
               c = read_point(startx, i); /* чтение точки */
               mempoint(x, j, c); /* запись ее в новое место */
          }
}
/* Вращение точки относительно специфицированной пользователем
   начальной точки x_org и y_org на угол theta */
void rotate_point(theta, x, y, x_org, y_org)
double theta, *x, *y;
int x_org, y_org;
{
     double tx, ty;
     /* нормализация координат x и y исходной точки */
     tx = *x - x_org;
     ty = *y - y_org;
     /* вращение */
     *x = tx * cos(theta) - ty * sin(theta);
```

```
     *y = tx * sin(theta) + ty * cos(theta);
     /* возврат значений координат */
     *x += x_org;
     *y += y_org;
}
/* Вращение специфицированного объекта */
void rotate_object(ob, theta, x, y, sides)
double ob[][4]; /* определение объекта */
double theta; /* угол поворота в радианах */
int x, y; /* центр вращения */
int sides;
{
     register int i, j;
     double tempx, tempy; /* это вспомогательные переменные,
               используемые при преобразовании типов */
     char ch;
     for(;;){
          ch = getch(); /* просматривается, если указывается
                           направление врашения */
          switch(tolower(ch)) {
               case 'l': /* вращение против часовой стрелки */
                    theta = theta < 0 ? -theta : theta;


                    break;
               case 'r': /* вращение по часовой стрелке */
                    theta = theta > 0 ? -theta : theta;
                    break;
               default: return;
          }
          for(j=0; j<sides; j++) {
               /* стирание предыдущей линии */
               line((int) ob[j][0], (int) ob[j][1],
                    (int) ob[j][2], (int) ob[j][3], 0);
               rotate_point(theta, &ob[j][0], &ob[j][1], x, y);
               rotate_point(theta, &ob[j][2], &ob[j][3], x, y);
               line((int) ob[j][0], (int) ob[j][1],
                    (int) ob[j][2], (int) ob[j][3], 2);
          }
     }
}
/* отображение объекта */
void display_object(ob, sides)
double ob[][4];
int sides;
{
     register int i;
     for(i=0; i<sides; i++)
          line((int) ob[i][0], (int) ob[i][1],
               (int) ob[i][2], (int) ob[i][3], 2);
}
/* Определение объекта по его габаритным точкам с помощью
   клавиатуры или "мыши" */
define_object(ob, x, y)
double ob[][4];
int x, y;
{
     union k{
          char c[2];
          int i;
     } key;
     register int i, j;
     char far *ptr = (char far *) 0xB8000000; /* Указатель на
                                                 CGA-память */
```

```
     char far *temp;
     unsigned char buf[14][80];
     int sides=0;
     int deltax, deltay, oldx, oldy;
     temp = ptr;


     /* сохранение верхней части текущего экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
               buf[i][j] = *temp; /* четный байт */
               buf[i][j+1] = *(temp+8152); /* нечетный байт */
               *temp = 0; *(temp+8152) = 0; /* очистка верхней
                                               части экрана */
               temp++;
          }
     i = 0;
     key.i = 0;
     xhairs(x, y);
     do {
          goto_xy(0, 0);
          printf("Определите сторону %d", sides+1);
          if(i==0) printf("Укажите первую габаритную точку");
          else printf("Укажите вторую габаритную точку");
          do {
/**************** Добавочная часть для мыши ******************/
          /* Просматривается если "мышь" перемещается */
            mouse_motion(&deltax, &deltay);
            /* используйте левую клавишу для определения точки*/
            if(leftb_pressed()) {
              xhairs(x, y); /* стирание графического курсора */
              /* запоминание координат точки */
              ob[sides][i++] = (double) x;
              ob[sides][i++] = (double) y;
              if(i==4) {
                   i = 0;
                   sides++;
              }
              break;
            }
          } while(!kbhit() && !deltax && ! deltay);
          if(leftb_pressed()) wait_on(LEFTB);
          if(deltax || deltay) {
            /* если "мышь" переместилась, то пересчет координат*/
            oldx = x; oldy = y;
            mouse_position(&y, &x);
            y = y / 2; /* нормализация координат виртуального
                          экрана*/
            /* стирание графического курсора */
            xhairs(oldx, oldy);
          }
/*********** Конец добавочной части для "мыши" *****************/
          else if(kbhit()) {
               key.i = bioskey(0);
               /* изображение графического курсора */
               xhairs(x, y);
               if(key.c[0]==13) {


                    /* используйте <ВВОД> для определения точки*/
                    ob[sides][i++] = (double) x;
                    ob[sides][i++] = (double) y;
                    if(i==4) {
                         i = 0;
```

```
                    sides++;
               }
          }
          /* если клавиши управления курсором, то
             перемещение графического курсора */
          if(!key.c[0]) switch(key.c[1]) {
               case 75: /* влево */
                    y-=1;
                    break;
               case 77: /* вправо */
                    y+=1;
                    break;
               case 72: /* вверх */
                    x-=1;
                    break;
               case 80: /* вниз */
                    x+=1;
                    break;
               case 71: /* влево вверх */
                    x-=1; y-=1;
                    break;
               case 73: /* вправо вверх */
                    x-=1; y+=1;
                    break;
               case 79: /* влево вниз */
                    x+=1; y-=1;
                    break;
               case 81: /* вправо вниз */
                    x+=1; y+=1;
                    break;
          }
     }
     if(key.c[1] != 59) xhairs(x, y);
} while(key.c[1] != 59); /* нажмите F1 для остановки */
temp = ptr;
/* восстановление изображения в верхней части экрана */
for(i=0; i<14; i++)
     for(j=0; j<80; j+=2) {
          *temp = buf[i][j];
          *(temp+8152) = buf[i][j+1];
          temp++;
     }
return sides;
}
/*  Отображение меню */
menu()


{
     register int i, j;
     char far *ptr = (char far *) 0xB8000000; /* Указатель на
                                                 CGA-память */
     char far *temp;
     unsigned char buf[14][80]; /* для хранения содержимого
                                   экрана */
     int x, y, choice;
     char items[][20] = {
          "BOX",
          "CIRCLE",
          "LINE",
          "FILL BOX",
          "FILL CIRCLE"
     };
     temp = ptr;
```

```
     /* сохранение верхней части текущего экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
               buf[i][j] = *temp; /* четный байт */
               buf[i][j+1] = *(temp+8152); /* нечетный байт */
               *temp = 0; *(temp+8152) = 0; /* очистка верхней
                                               части экрана */
               temp++;
          }
     goto_xy(0, 0);
     /* ожидание, которое будет прервано в результате нажатия на
        клавишу */
     while(rightb_pressed() || leftb_pressed());
     cursor_on();
     choice = mouse_menu(5, items, 0, 0);
     cursor_off();
     temp = ptr;
     /* восстановление изображения верхней части экрана */
     for(i=0; i<14; i++)
          for(j=0; j<80; j+=2) {
               *temp = buf[i][j];
               *(temp+8152) = buf[i][j+1];
               temp++;
          }
     return choice;
}
/*********************************************************/
/* Функции, обеспечивающие интерфейс с "мышью"           */
/*********************************************************/
/*  Включение курсора "мыши"  */


void cursor_on()
{
     int fnum;
     fnum = 1; /* отобразить курсор */
     cmouses( &fnum,  &fnum,  &fnum,  &fnum);
}
/*  Выключение курсора "мыши"  */
void cursor_off()
{
     int fnum;
     fnum = 2; /* стереть курсор */
     cmouses( &fnum,  &fnum,  &fnum,  &fnum);
}
/* Возвращает значение "истина", если нажата правая клавиша,
     и "ложь" в противном случае                          */
rightb_pressed()
{
     int fnum, arg2, arg3, arg4;
     fnum = 3;   /* Чтение позиции и статуса клавиши   */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     return arg2 & 2;
}
/* Возвращает значение "истина", если нажата левая клавиша,
     и "ложь" в противном случае */
leftb_pressed()
{
     int fnum, arg2, arg3, arg4;
     fnum = 3;   /* Чтение позиции и статуса клавиши   */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     return arg2 & 1;
}
/* Установить координаты курсора "мыши" */
```

```
void set_mouse_position(x, y)
int x, y;
{
     int fnum, arg2;
     fnum = 4; /* установка позиции */


     cmouses(&fnum, &arg2, &x, &y);
}
/* Возвращает координаты курсора "мыши" */
void mouse_position(x, y)
int *x, *y;
{
     int fnum, arg2, arg3, arg4;
     fnum = 3; /* получить позицию и статус клавиши */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     *x = arg3;
     *y = arg4;
}
/*  Возвращает направление движения  */
void mouse_motion(deltax, deltay)
char *deltax, *deltay;
{
     int fnum, arg2, arg3, arg4;
     fnum = 11; /* получить направление движения */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     if(arg3>0) *deltax = RIGHT;
     else if(arg3<0) *deltax = LEFT;
     else *deltax = NOT_MOVED;
     if(arg4>0) *deltay = DOWN;
     else if(arg4<0) *deltay = UP;
     else *deltay = NOT_MOVED;
}
/* Отображает однострочное меню для "мыши" и возвращает
     код выбранного пользователем элемента меню */
mouse_menu(count, item, x, y)
int count;           /* количество элементов меню */
char item[][20];      /* элементы меню */
int x, y;           /* позиции отображения */
{
     int i, len[MENU_MAX][2], t;
     int mousex, mousey;
     goto_xy(x, y);
     t = 0;
     for(i=0; i<count; i++) {
          printf("%s   ", item[i]);
          len[i][0] = t;
          /* каждый символ имеет ширину в 16 точек растра */
          len[i][1] = t + strlen(item[i])*16;
          t = len[i][1] + 32; /* добавляется два пробела между
                                 элементами меню */
     }
     /* ожидание выбора пользователем элемента меню */


     do {
          if(rightb_pressed() || leftb_pressed()) break;
     } while(!kbhit());
     /* ожидание нажатия клавиши */
     while(rightb_pressed() || leftb_pressed());
     /* получить текущую позицию курсора "мыши" */
     mouse_position(&mousex, &mousey);
     /* анализируется, находится ли курсор в пределах меню */
          if(mousey>=0 && mousey<8)  /* символ имеет высоту
```

```
                              8 точек растра */
               for(i=0; i<count; i++) {
                    if(mousex>len[i][0] && mousex<len[i][1])
                         return i;
               }
          return i;
     }
/* Возвращает 1, если специфицированная клавиша не нажата */
void wait_on(button)
int button;
{
     if(button== LEFTB)
          while(leftb_pressed());
     else
          while(rightb_pressed());
}
/* Установка "мыши" в исходное состояние   */
void mouse_reset()
{
     int fnum, arg2, arg3, arg4;
     fnum = 0;  /* Установка "мыши" в исходное состояние  */
     cmouses( &fnum, &arg2, &arg3, &arg4);
     if(fnum!=-1) {
     printf("Аппаратные или программные средства поддержки ");
     printf("'мыши' не инсталированы");
          exit(1);
     }
     if(arg2!=2) {
   printf("Разрешено использование только двухклавишной 'мыши'");
          exit(1);
     }
}
```

Программа рисования, предложенная вам в этой главе, довольно удобна в обращении, так как позволяет вам рисовать произвольные рисунки (естественно используя стандартные линии), двигаясь при этом в любых произвольных направлениях по экрану. Например, рисунок 9.1 был нарисован на экране дисплея буквально за несколько минут. Рис. 9.2 иллюстрирует, что изображается на экране при вызове меню "мыши" (форма курсора - стрелка принята по умолчанию).

НЕКОТОРЫЕ ВОЗМОЖНОСТИ РАСШИРЕНИЯ ВЫПОЛНЯЕМЫХ ФУНКЦИЙ ПРОГРАММЫ

Во-первых, вам возможно понадобится модифицировать функцию mouse_menu() с тем, чтобы возможно было обрабатывать меню, занимающее более одной строки экрана. Фактически для реализации этого наиболее приемлемым путем будет объединение функции обработки меню "мыши" с функциями обработки "всплывающего" меню, рассмотренными в главе 1.

Если вы привыкли работать с иконным интерфейсом, то вы, в принципе, можете модифицировать функции обработки меню "мыши" таким образом, чтобы вместо слов в меню фигурировали на соответствующих местах пиктограммы, обозначающие тот или иной элемент меню. Возможно также модифицировать функции таким образом, чтобы обеспечить полиэкраный режим работы "мыши", а также копирование (а не перемещение) части экрана (или изображения) с одного места в другое.

СОЗДАНИЕ КОММЕРЧЕСКИХ ДИАГРАММ

В данной главе мы разработаем и опишем программы, которые могут быть использованы для создания наиболее удобного вида представления коммерческих данных - диаграмм. Возможность получить числовую информацию в визуальной форме часто оказывается очень полезной. Как вы вскоре убедитесь, создание диаграмм не представляет таких трудностей, как это первоначально кажется.

Глава X начинается описанием простого инструментария, позволяющего строить диаграммы. Вторая часть главы содержит описания, как этот инстументарий может быть использован для построения простых, но полезных программ, которые позволяют выводить несколько диаграмм одновременно.

Примеры в главе написаны для компьютеров семейства PC с цветным графическим адаптером. Используется четвертый видеорежим, так как он поддерживает все виды цветных адаптеров. Однако, вы можете легко изменить предложенные функции для того, чтобы работать с другими типами адаптеров.

Глава X
НОРМАЛИЗАЦИЯ ДАННЫХ

Перед разработкой программы отображения данных на экране вам следует уяснить, как численные значения переводятся в соответствующие координаты экрана. Как вы помните, размерность экрана в четвертом видеорежиме 320*200, причем 320 - горизонтальная размерность и 200 - вертикальная. Учитывая, что диаграммы изображаются вертикальными полосами, данные должны быть преобразованы таким образом, чтобы они принимали значения в диапазоне от 0 до 199. Данный процесс преобразования называется нормализацией.

Чтобы нормализовать значение, необходимо умножить его на некоторый коэффициент, гарантирующий получение результата в диапазоне размера экрана. Для определения коэффициента, необходимо знать максимальное и минимальное значения чисел, выводимых в виде диаграммы. Для определения подходящего коэффициента, необходимо вычесть минимальное значение из максимального и поделить вертикальную размерность экрана на полученную разность. Иными словами, для 4-го видеорежима нормирующий множитель определяется по формуле:

```
нормирующий_множитель = 200 / (max - min)
```

Таким образом, каждый элемент данных нормализуется по формуле:

```
нормализованное_данное = необработанное_данное * норм_множитель
```

Глава X
РАЗРАБОТКА ФУНКЦИЙ ПОСТРОЕНИЯ ДИАГРАММ

Прежде, чем разрабатывать функцию, рисующую диаграммы, необходимо точно определить, что она будет делать. Во-первых, она должна выполнять свою главную задачу - выводить данные в виде диаграмм. Функция должна допускать использование в качестве входного параметра массива чисел с плавающей точкой и преобразовывать их в нормализованные целые эквиваленты. Программа должна быть реентерабельной и позволять рисовать несколько диаграмм одновременно. Функция должна также содержать аргумент, определяющий расстояние между диаграммами, соответствующими разным последовательностям данных, и, наконец, она должна позволять определять толщину линий диаграммы.

Программа функции bargraph(), приведенная ниже, удовлетворяет этим требованиям.

```
     /* Вывод диаграммы */
void bargraph(data,num,offset,min,max,width)
double *data;       /* массив данных */
int num;            /* количество элементов в массиве */
int offset;         /* расстояние между диаграммами */
int min,max;        /* мин. и мак. выводимые значения */
```

```
int width;          /* толщина линий */
{
     int y,t,incr;
     double norm_data,norm_ratio,spread;
     char s[80];
     static int color = 0;
     int tempwidth;
     /* всегда используйте различные цвета */
     color++;
     if( color > 3 ) color = 1;
     /* определение нормирующего множителя */
     spread = (double)max-min;
     norm_ratio = 180/spread;
     incr = 280/num;/* определение промежутка между значениями */
     tempwidth = width;
     for (t=0;t<num;++t) {
          norm_data = data[t];
     /* подгонка отрицательных значений */
          norm_data = norm_data-(double)min;
          norm_data *= norm_ratio; /* нормирование */
          y = (int)norm_data; /* преобразование типа */
          do {
               Line(179,((t*incr)+20+offset+width),179-y,
                 ((t*incr)+20+offset+width),color);
          width--;
                              Глава X
          } while(width);
          width = tempwidth;
          }
     }
```

Давайте тщательно разберем данную программу. Функция bargraph() получает через входные параметры: массив данных, число элементов в массиве, расстояние между диаграммами (для случая одновременного вывода нескольких диаграмм), минимальное и максимальное значения данных и ширину линий диаграмм (ширина линии задается в единицах растра). Статическая переменная color определяет новый цвет при повторных обращениях к bargraph(). Таким образом, различные последовательности данных при их одновременном выводе будут изображены диаграммами различного цвета. При вычислении нормирующего множителя вместо максимальной высоты экрана (200 для 4-го видеорежима) использовано меньшее число - 180, что в последующем позволит использовать две строки экрана для вывода поясняющей информации. Обычно удобнее, если диаграмма полностью занимает экран независимо от количества выводимых чисел. Например, диаграмма, отражающая малые наборы данных, выглядит более привлекательной, если она занимает весь экран, а не совокупность сбившихся в кучу вертикальных полос в одном из углов экрана. Для размещения диаграммы относительно ширины экрана последняя (здесь также целесообразнее использовать меньшее число 280 вместо 300) делится на количество выводимых элементов, полученный результат затем используют при определении горизонтальных координат стержней диаграммы. В конце программы выполняется циклическая нормализация данных и вычерчивание линий заданной толщины с указанным смещением.

Функция bargraph() - ключевая функция, но это только одно из многих средств, позволяющих вам рисовать диаграммы почти любого вида. Основные из этих средств вы узнаете в процессе дальнейшего изложения материала.

Вычерчивание линии нулевого уровня.

Диаграмма будет выглядеть более привлекательной и наглядной, если вдоль нижнего края вычертить линию нулевого уровня. Программа функции grid(), представленная в данном разделе, служит именно для этих целей.

```
     /* Вывод линии нулевого уровня */
void grid(min,max)
int min,max;
{
     register int t;
     goto_xy(22,0); printf("%d",min);
     goto_xy(0,0); printf("%d",max);
     line(180,10,180,300,1);
}
```


Вы видите, что функция grid() так же, как и bargraph() оставляет внизу две строки для вывода поясняющих меток и другой справочной информации.

Вывод меток элементов диаграмм.

Часто пользователю необходимо помечать значения, выводимые диаграммой. Например, на диаграмме, показывающей изменение прибыли корпорации за пять лет, целесообразно каждый стержень диаграммы пометить соответствующим годом. Конечно, вы всегда можете это сделать вручную, используя функции **goto**_xy() и printf(); функция label(), представленная ниже, освободит вас от этой рутинной работы, она автоматически выводит необходимые метки в нужном месте экрана. Входными параметрами функции label() являются: массив меток и их количество. Длина каждой метки ограничена 20 символами (включая указатель конца), но это не является жестким ограничением и при небходимости вы можете легко изменить максимальную длину меток.

```
          /* Вывод меток на экран */
void label(str,num)
char str[][20]; /* массив меток */
int num; /* количество меток */
{
     int i,j,inc;
     inc = 38/num;
     i = 2; /* определение начальной точки */
     for (j=0;j<num;j++) {
          goto_xy(23,i);
          printf(str[j]);
          i += inc;
     }
}
```

Вычерчивание вспомогательных линий.

В некоторых случаях полезно выводить горизонтальные полосы для сравнения высот стержней диаграммы. Так как сплошные линии могут отвлекать пользователя, то для этой цели лучше использовать пунктирные линии. Функция hashlines(), приведенная ниже, рисует требуемые пунктирные линии.

```
          /* Вывод пунктирных линий */
void hashlines()
```



```
{
     int i,j;
     for (i=10;1<180;i+=10) {
          for (j=10;j<300;j+=5)
               mempoint(i,j,3); /* одна точка на каждые 5 единиц
                                   растра */
     }
}
```

Вывод надписей.

При одновременном выводе нескольких наборов в виде диаграмм полезно определить цвет диаграммы, соответствующий каждому набору. Это можно сделать, например, выдав надпись, содержащую

наименование набора и используемый для него цвет диаграммы. Функция legend(), приведенная здесь, выводит наименования наборов и прямоугольник соответствующего цвета, в качестве входных параметров она использует список наименований и их количество. Функция legend() использует функцию fill_box(), описанную ранее, для вывода цветного прямоугольника.

```
          /* Вывод надписи */
void legend(names,num)
char names[][20];
int num;            /* количество наименований */
{
     int color = 1,i,j;
     goto_xy(24,0); /* надпись производится в последней строке */
     j = 0;
     for (i=0;i<num;i++) {
          /* вывод наименования */
          printf("%s   ",names[i]);
          /* определение координаты цветного прямоугольника. В 4
             режиме каждому литерному символу отводится 8 единиц
             растра (в ширину)  */
          j += strlen(names[i]) * 8 + 4;
          fill_box(192,j,198,j+12,color);
          j += 28; /* продвижение к следующему полю вывода */
          color ++;
          if( color>3 ) color = 1;
     }
}
```

Графический рисунок на стр 355 не может быть воиспроизведен имеющимися средствами. (Ред. перевода И.Бычковский.)

Рис.10-1. Результат работы программы построения диаграмм

Глава X

Простейшая программа вывода диаграмм.

Следующая программа показывает все описанные функции в действии. Результат ее работы представлен на рис.10-1. Программа выводит среднюю стоимость акций трех мнимых корпораций за пять лет.

```
     /* Программная демонстрация построения диаграмм */
#include "dos.h"
void bargraph(),mode(),mempoint();
void line(),goto_xy(),grid(),label();
void hashlines(),legend(),read_cursor_xy();
void palette(),color_puts(),fill_box();
main()
{
     double widget[] = {
          10.1,20,30,35.34,50
          };
     double global[] = {
          19,20,8.8,30,40
          };
     double tower[] = {
          25.25,19,17.4,33,29
          };
     int min,max;
     char n[][20] = {
          "widget",
          "global",
          "tower"
          };
     char lab[][20] = {
          "1983",
```

```
          "1984",
          "1985",
          "1986",
          "1987"
          };
     mode(4);       /* выбор режима 320*200 */
     palette(0);
     grid(0,50);    /* построение линии нулевого уровня */
     hashlines();   /* вывод пунктирных линий */
     label(lab,5);  /* вывод чисел */
     legend(n,3);   /* вывод надписей */
     /* вывод курса акций трех кампаний */
     bargraph(widget,5,0,0,50,4);
     bargraph(global,5,10,0,50,4);
     bargraph(tower,5,20,0,50,4);
                          Глава X
     getch();
     mode(3);
}
/* Вывод линии нулевого уровня диаграммы */
void grid(min,max)
int min,max;
{
     register int t;
     goto_xy(22,0); printf("%d",min);
     goto_xy(0,0); printf("%d",max);
     line(180,10,180,300,1);
}
/* вывод меток на экран */
void label(str,num)
char str[][20]; /* массив меток */
int num; /* количество меток */
{
     int i,j,inc;
     inc = 38/num;
     i = 2; /* определение начальной точки */
     for (j=0;j<num;j++) {
          goto_xy(23,i);
          printf(str[j]);
          i += inc;
     }
}
/* Вывод пунктирных линий на экран */
void hashlines()
{
     int i,j;
     for (i=10;1<180;i+=10) {
          for (j=10;j<300;j+=5)
               mempoint(i,j,3); /* одна точка на каждые 5 единиц
                                   растра */
     }
}
/* вывод надписи */
void legend(names,num)
char names[][20];
int num; /* количество наименований */
{
     int color = 1,i,j;
     goto_xy(24,0); /* надпись производится в последней строке */
     j = 0;
     for (i=0;i<num;i++) {
          /* вывод наименования */
                          Глава X
          printf("%s   ",names[i]);
```

```
          /* определение координаты цветного прямоугольника. В 4
             режиме каждому литерному символу отводится 8 единиц
             растра ( в ширину )  */
          j++ = strlen(names[i]*8+4);
          fill_box(192,j,198,j+12,color);
          j++ = 28; /* продвижение к следующему полю вывода */
          color ++;
          if( color>3 ) color = 1;
     }
}
/* Вычерчивание диаграммы */
void bargraph(data,num,offset,min,max,width)
double *data; /* массив данных */
int num; /* количество элементов в массиве */
int offset; /* расстояние между диаграммами */
int min,max; /* минимальное и максимальное выводимые значения */
int width; /* толщина линий */
{
     int y,t,incr;
     double norm_data,norm_ratio,spread;
     char s[80];
     static int color = 0;
     int tempwidth;
     /* всегда используйте различные цвета */
     color++;
     if( color > 3 ) color = 1;
     /* определение нормирующего множителя */
     spread = (double)max-min;
     norm_ratio = 180/spread;
     incr = 280/num; /* определение промежутка между значениями*/
     tempwidth = width;
     for (t=0;t<num;++t) {
          norm_data = data[t];
     /* подгонка отрицательных значений */
          norm_data = norm_data-(double)min;
          norm_data *= norm_ratio; /* нормирование */
          y = (int)norm_data; /* преобразование типа */
          do {
               line(179,((t*incr)+20+offset+width),179-y,
                 ((t*incr)+20+offset+width),color);
          width--;
          } while(width);
          width = tempwidth;
          }
     }
/* Вывод линии заданного цвета, используя базовый алгоритм
                              Глава X
     Брезенхама */
void line(startx,starty,endx,endy,color)
int startx,starty,endx,endy,color;
{
     register int t,distance;
     int x=0,y=0,delta_x,delta_y;
     int incx,incy;
/* вычисление расстояний по обоим направлениям */
     delta_x = endx - startx;
     delta_y = endy - starty;
/* определение направлений увеличения координат, нулевое
   увеличение соответствует либо вертикальной, либо
   горизонтальной линии */
     if( delta_x > 0 ) incx = 1 ;
     else  if(delta_x == 0 ) incx = 0;
     else incx = -1;
     if( delta_y > 0 ) incy = 1 ;
```

```
     else  if(delta_y == 0 ) incy = 0;
     else incy = -1;
/* определение максимума изменения координат */
     delta_x = abs(delta_x);
     delta_y = abs(delta_y);
     if( delta_x > delta_y ) distance = delta_x;
     else distance = delta_y;
/* вычерчивание линии */
     for (t=0;t<=distance+1;t++) {
          mempoint(startx,starty,color);
          x+= delta_x;
          y+= delta_y;
          if(x>distance) {
          x-=distance;
          startx+=incx;
          }
          if(y>distance) {
          y-=distance;
          starty+=incy;
          }
     }
}
/* наполнение прямоугольника заданным цветом */
void fill_box(startx,starty,endx,endy,color_code)
int startx, starty, endx, endy, color_code;
{
     register int i,begin,end;
     begin = startx < endx ? startx : endx;
     end = startx > endx ? startx : endx;
     for (i=begin;i<=end;i++)
          line(i,starty,i,endy,color_code);
}
/* запись точки в CGA/EGA память */
void mempoint(x,y,color_code)
                              Глава X
int x,y,color_code;
{
     union mask {
          char c[2];
          int i;
     } bit_mask;
     int i,index,bit_position;
     unsigned char t;
     char xor; /* xor - цвет или наложение */
     char far *ptr = (char far *) 0xB8000000; /* указатель на
                                                 CGA  */
     bit_mask.i = 0xFF3F; /* 11111111 00111111 в двоичном коде */
     /* контроль координат для 4 режима */
     if(x<0 || x>199 || y<0 || y>319) return;
     xor = color_code & 128; /* проверка установки режима xor */
     color_code = color_code & 127; /* маска 7 старших бит */
/* установка bit_mask и color_code в правильное положение */
     bit_position = y%4;
     color_code <<= 2*(3-bit_position);
     bit_mask.i >>= 2*bit_position;
/* поиск соответствующего байта в памяти экрана */
     index = x*40 + (y>>2);
     if(x%2) index+=8152; /* если нечетный, использовать второй
                             байт */
/* запись цвета */
     if(!xor) { /* режим наложения */
          t = *(ptr + index) & bit_mask.c[0];
          *(ptr + index) = t | color_code;
     }
```

```
     else { /* режим xor */
           t = *(ptr + index) | (char)0;
          *(ptr + index) = t | color_code;
     }
}
/* установка видеорежима */
void mode(mode_code)
int mode_code;
{
     union REGS r;
     r.h.al = mode_code;
     r.h.ah = 0;
     int86(0x10,&r,&r);
}
/* установка курсора в координаты x,y */
void goto_xy(x,y)
int x,y;
{
     union REGS r;
```



```
     r.h.ah = 2; /* функция адресации курсора */
     r.h.dl = y; /* горизонтальная координата */
     r.h.dh = x; /* вертикальная координата */
     r.h.bh = 0; /* видеостраница */
     int86(0x10,&r,&r);
}
/* установка цветов диаграмм */
void palette(pnum)
int pnum;
{
     union REGS r;
     r.h.bh = 1; /* код 4 режима */
     r.h.bl = pnum;
     r.h.ah = 11; /* установка функции цвета */
     int86(0x10,&r,&r);
}
```

Глава X

ПРОГРАММА ВЫЧЕРЧИВАНИЯ ДИАГРАММ

Вы можете использовать описанные функции для построения программы создания диаграмм. Программа позволяет пользователю вводить количество наборов данных, количество элементов в каждом наборе, наименования и метки соответствующих данных, а также толщину линий и расстояния между диаграммами. После ввода указанных данных программа автоматически вычерчивает диаграмму. Вы также можете написать программу сохранения построенной диаграммы в файле для ее дальнейшего использования.

Главная программа.

Здесь приводится основная функция main(), описывающая алгоритм построения диаграмм и содержащая несколько макросов.

```
#define MAX_SETS 3
#define MAX_ENTRIES 50
#define MAX_LABELS 20
#define MAX_NAMES 20
main()
{
     double v[MAX_SETS][MAX_ENTRIES]; /* размещение данных */
     int num_entries;
     int num_sets;
     int min,max,i;
     int lines,offset;
     char save = 0; /* признак сохранения диаграммы */
     char names[MAX_NAMES][20];
```

```
     char lab[MAX_LABELS][20];
/* считывание данных */
     enter(v,&num_entries,&num_sets);
/* поиск минимального и максимального значения */
     min_max(v,num_entries,num_sets,&min,&max);
/* ввод наименований данных */
     get_names(names,num_sets);
/* ввод меток для диаграммы */
     get_labels(lab,num_entries);
/* ввод толщины линии */
     lines = get_line_size();
/* ввод интервала между диаграммами */
     offset = get_offset();
```



```
/* сохранить диаграмму в файле ? */
     printf(" сохранить диаграмму в файле ? (y/n) ");
     if(tolower(getche()) == 'y') save = 1;
     mode(4); /* графический режим 320*200 */
     palette(0);
     grid(min,max); /* вывод линии нулевого уровня */
     hashlines(); /* вывод пунктирных линий */
     label(lab,num_entries); /* вывод меток диаграммы */
     legend(names,num_sets); /* вывод пояснительных надписей */
     /* вывод значений в виде диаграммы */
     for (i=0;i<num_sets;i++)
          bargraph(v[i],num_entries,i*offset,min,max,lines);
     if(save) save_pic();
     getch();
     mode(3);
}
```

Как вы видите, функция main() начинается описанием переменных, значения которых устанавливает пользователь. Массив v определен достаточно большим, чтобы содержать до трех наборов данных до 50 элементов каждый. (Эти размеры являются произвольными и при желании вы можете их изменить.) Затем функция считывает выводимые пользователем в виде диаграмм данные и определяет минимальное и максимальное значение данных. После этого на экран выводятся линия нулевого уровня, пунктирные линии уровня, метки диаграммы и наименование наборов. В завершение вычерчивается сама диаграмма. Перед выходом происходит сохранение диаграммы при помощи функции save_pic(). Давайте рассмотрим некоторые используемые в программе main() функции, которые не входят в описанные выше инструментарии построения диаграмм.

Функция enter().

Приведенная здесь функция enter() использует в качестве своих параметров адрес массива, в котором будут размещены данные, и адреса переменных для размещения числа количества элементов в наборе и числа самих наборов. Функция начинает свою работу с запроса у пользователя количества наборов данных и затем количества элементов данных в каждом наборе. После получения этой информации производится считывание данных для каждого набора.

```
     /* Считывание данных */
enter(v,entries,sets)
double v[][MAX_ENTRIES]; /*  массив данных */
int *entries; /* количество элементов в каждом наборе данных */
int *sets; /* количество наборов данных */
{

     int i,j,count,num;
     char s[80];
printf("Введите число наборов данных (от 1 до %d)",MAX_SETS);
     scanf("%d%c",&count,&j);
     if(count>MAX_SETS) count = MAX_SETS; /* выход за границы
```

```
                                                  массива */
     *sets = count;
printf("Введите количество элементов (от 1 до %d) ",MAX_ENTRIES);
     scanf("%d%c",&num,&j);
     if(num>MAX_SETS) num = MAX_ENTRIES; /* выход за границы
                                                  массива */
     *entries = num;
     j = 0;
     /* считывание значений данных */
     while((j<count)) {
          printf("Набор данных %d\n",j+1);
          for (i=0;i<num;i++) {
               printf("%d:",i+1);
               gets(s);
               sscanf(s,"%lf",&v[j][i]);
          }
          j++;
     }
     return count;
}
```

Функция min_max().

Так как функция bargraph() использует максимальное и минимальное значения выводимых данных, то нам потребуется специальная функция для определения этих значений. Необходимо также отметить, что эта функция должна не просто определять минимальное и максимальное значения набора данных, а находить наименьшее минимальное и наибольшее максимальное значения для нескольких наборов данных, что обеспечит соответствие при одновременном построении сразу нескольких диаграмм. Функция min_max(), приведенная здесь, вместе с двумя внутренними функциями удовлетворяет этому требованию.

```
     /* Поиск наименьшего минимума и наибольшего максимума
        среди всех наборов данных */
void min_max(v,entries,sets,min,max)
double v[][MAX_ENTRIES]; /*  значения */
int entries; /* количество входов для каждого набора
                данных */
int sets; /* количество наборов данных */
int *min,*max; /* возвращает минимальное и максимальное
                  значение */

{
     int i,j;
     int tmin,tmax;
     *min = *max = 0;
     for (i=0;i<sets;i++) {
          tmax = getmax(v[i],entries);
          tmin = getmin(v[i],entries);
          if(tmax>*max) *max = tmax;
          if(tmin <*min) *min = tmin;
     }
}
 /* Возврат максимального значения данных */
getmax(data,num)
double *data;
int num;
{
     int t,max;
     max = (int)data[0];
     for (t=1;t<num;++t)
          if(data[t]>max) max = (int)data[t];
     return max;
}
```

```
/* Возврат минимального значения данных */
getmin(data,num)
double *data;
int num;
{
     int t,min;
     min = (int)data[0];
     for (t=1;t<num;++t)
          if(data[t]<min) min = (int)data[t];
     return min;
}
```

Полный текст программы вычерчивания диаграмм.

Полный текст программы вычерчивания диаграмм представлен ниже.

```
     /* Программа генерации диаграмм */
#include "dos.h"
#include "stdio.h"
#define MAX_SETS 3
#define MAX_ENTRIES 50

#define MAX_LABELS 20
#define MAX_NAMES 20
void bargraph(),mode(),mempoint();
void line(),goto_xy(),grid(),label();
void hashlines(),legend(),read_cursor_xy();
void palette(),color_puts(),fill_box();
void get_labels(),get_names(),min_max();
void save_pic();
main()
{
     double v[MAX_SETS][MAX_ENTRIES]; /* размещение данных */
     int num_entries;
     int num_sets;
     int min,max,i;
     int lines,offset;
     char save = 0; /* признак записи диаграммы */
     char names[MAX_NAMES][20];
     char lab[MAX_LABELS][20];
/* считывание данных */
     enter(v,&num_entries,&num_sets);
/* поиск минимального и максимального значения */
     min_max(v,num_entries,num_sets,&min,&max);
/* ввод наименований данных */
     get_names(names,num_sets);
/* ввод меток для диаграммы */
     get_labels(lab,num_entries);
/* ввод толщины линии */
     lines = get_line_size();
/* ввод интервала между диаграммами */
     offset = get_offset();
/* сохранить диаграмму в файле ? */
     printf(" сохранить диаграмму в файле ? (y/n) ");
     if(tolower(getche()) == 'y') save = 1;
     mode(4); /* графический режим 320*200 */
     palette(0);
     grid(min,max); /* вывод линии нулевого уровня */
     hashlines(); /* вывод пунктирных линий */
     label(lab,num_entries); /* вывод меток диаграммы */
     legend(names,num_sets); /* вывод пояснительных надписей */
     /* вывод значений в виде диаграммы */
     for (i=0;i<num_sets;i++)
          bargraph(v[i],num_entries,i*offset,min,max,lines);
```

```
     if(save) save_pic();
     getch();
     mode(3);
}
     /* считывание данных */
enter(v,entries,sets)
double v[][MAX_ENTRIES]; /*  массив данных */
int *entries;      /* количество элементов данных в каждом наборе
                      данных */
int *sets; /* количество наборов данных */
{
     int i,j,count,num;
     char s[80];
printf("Введите число наборов данных (от 1 до %d)",MAX_SETS);
     scanf("%d%c",&count,&j);
     if(count>MAX_SETS) count = MAX_SETS; /* выход за границы
                                             массива */
     *sets = count;
printf("Ведите число элементов данных (от 1 до %d)",MAX_ENTRIES);
     scanf("%d%c",&num,&j);
     if(num>MAX_ENTRIES) num = MAX_ENTRIES; /* выход за границы
                                              массива */
     *entries = num;
     j = 0;
     /* считывание значений */
while((j<count)) {
     printf(" Набор данных %d\n",j+1);
     for (i = 0;i<num;i++) {
     printf("%d:",i+1);
     gets(s);
     sscanf(s,"%lf",&v[j][i]);
     }
     j++;
   }
   return count;
}
/* Ввод имен наборов */
void get_names(n,num)
char n[][20];         /* массив для имен */
int num;              /* число наборов */
{
     int i;
     for (i=0;i<num;i++) {
          printf(" Введите имя: ");
          gets(n[i]);
     }
}

/* Ввод метки каждого входа */
void get_labels(l,num)
char l[][20]; /* массив для меток */
int num; /* число входов */
{
     int i;
     for (i=0;i<num;i++) {
          printf(" Введите имя метки: ");
          gets(l[i]);
     }
}
/* Ввод интервала между диаграммами в единицах растра */
get_offset()
{
     int i;
printf(" Введите интервал между диаграммами в единицах растра");
```

```
     scanf("%d%*c",&i);
     return i;
}
/* Ввод толщины диаграмм в единицах растра */
get_line_size()
{
     int i;
printf("Введите толщину диаграммы в единицах растра : ");
     scanf("%d",&i);
     return i;
}
/* Вывод линии нулевого уровня диаграммы */
void grid(min,max)
int min,max;
{
     register int t;
     goto_xy(22,0); printf("%d",min);
     goto_xy(0,0); printf("%d",max);
     line(180,10,180,300,1);
}
/* Вывод меток на экран */
void label(str,num)
char str[][20]; /* массив меток */
int num; /* количество меток */
{
     int i,j,inc;
     inc = 38/num;
     i = 2; /* определение начальной точки */
     for (j=0;j<num;j++) {
          goto_xy(23,i);

          printf(str[j]);
          i += inc;
     }
}
/* Вывод пунктирных линий на экран */
void hashlines()
{
     int i,j;
     for (i=10;1<180;i+=10) {
          for (j=10;j<300;j+=5)
               mempoint(i,j,3); /* одна точка на каждые 5 единиц
                                   растра */
     }
}
/* Вывод надписи */
void legend(names,num)
char names[][20];
int num; /* количество наименований */
{
     int color = 1,i,j;
     goto_xy(24,0); /* надпись производится в последней строке */
     j = 0;
     for (i=0;i<num;i++) {
          /* Вывод наименования */
          printf("%s   ",names[i]);
          /* определение координаты цветного прямоугольника. В 4
             режиме каждому литерному символу отводится 8 единиц
             растра ( в ширину )  */
          j += strlen(names[i]) * 8 + 4;
          fill_box(192,j,198,j+12,color);
          j += 28; /* продвижение к следующему полю вывода */
          color ++;
          if( color>3 ) color = 1;
```

```
     }
}
void bargraph(data,num,offset,min,max,width)
double *data;          /* массив данных */
int num;               /* количество элементов в массиве */
int offset;            /* расстояние между диаграммами */
int min,max;  /* минимальное и максимальное выводимые значения */
int width;             /* толщина линий */
{
     int y,t,incr;
     double norm_data,norm_ratio,spread;
     char s[80];
     static int color = 0;
     int tempwidth;
     /* всегда используйте различные цвета */

     color++;
     if( color > 3 ) color = 1;
     /* определение нормирующего множителя */
     spread = (double)max-min;
     norm_ratio = 180/spread;
     incr = 280/num; /* определение промежутка между значениями*/
     tempwidth = width;
     for (t=0;t<num;++t) {
          norm_data = data[t];
     /* подгонка отрицательных значений */
          norm_data = norm_data-(double)min;
          norm_data *= norm_ratio; /* нормирование */
          y = (int)norm_data; /* преобразование типа */
          do {
               Line(179,((t*incr)+20+offset+width),179-y,
                 ((t*incr)+20+offset+width),color);
          width--;
          } while(width);
          width = tempwidth;
          }
     }
     /* поиск наименьшего минимума и наибольшего максимума
        среди всех наборов данных */
void min_max(v,entries,sets,min,max)
double v[][MAX_ENTRIES]; /*  значения */
int entries; /* количество входов для каждого набора
                данных */
int sets; /* количество наборов данных */
int *min,*max; /* возвращает минимальное и максимальное
                  значение */
{
     int i,j;
     int tmin,tmax;
     *min = *max = 0;
     for (i=0;i<sets;i++) {
          tmax = getmax(v[i],entries);
          tmin = getmin(v[i],entries);
          if(tmax>*max) *max = tmax;
          if(tmin <*min) *min = tmin;
     }
}
 /* Возврат максимального значения данных */
getmax(data,num)
double *data;
int num;
{
     int t,max;
```

```
     max = (int)data[0];
     for (t=1;t<num;++t)
          if(data[t]>max) max = (int)data[t];
     return max;
}
/* Возврат минимального значения данных */
getmin(data,num)
double *data;
int num;
{
     int t,min;
     min = (int)data[0];
     for (t=1;t<num;++t)
          if(data[t]<min) min = (int)data[t];
     return min;
}
/* Вывод линии заданного цвета, используя базовый алгоритм
     Брезенхама */
void line(startx,starty,endx,endy,color)
int startx,starty,endx,endy,color;
{
     register int t,distance;
     int x=0,y=0,delta_x,delta_y;
     int incx,incy;
/* вычисление расстояний по обоим направлениям */
     delta_x = endx - startx;
     delta_y = endy - starty;
/* определение направлений увеличения координат, нулевое
   увеличение соответствует либо вертикальной, либо
   горизонтальной линии */
     if( delta_x > 0 ) incx = 1 ;
     else  if(delta_x == 0 ) incx = 0;
     else incx = -1;
     if( delta_y > 0 ) incy = 1 ;
     else  if(delta_y == 0 ) incy = 0;
     else incy = -1;
/* определение максимума изменения координат */
     delta_x = abs(delta_x);
     delta_y = abs(delta_y);
     if( delta_x > delta_y ) distance = delta_x;
     else distance = delta_y;
/* вычерчивание линии */
     for (t=0;t<=distance+1;t++) {
          mempoint(startx,starty,color);
          x+= delta_x;
          y+= delta_y;
          if(x>distance) {
          x-=distance;
          startx+=incx;

          }
          if(y>distance) {
          y-=distance;
          starty+=incy;
          }
     }
}
/* наполнение прямоугольника заданным цветом */
void fill_box(startx,starty,endx,endy,color_code)
int startx,starty,endx,endy,color_code;
{
     register int i,begin,end;
     begin = startx < endx ? startx : endx;
     end = startx > endx ? startx : endx;
```

```
     for (i=begin;i<=end;i++)
          line(i,starty,i,endy,color_code);
}
/* запись точки в CGA/EGA память */
void mempoint(x,y,color_code)
int x,y,color_code;
{
     union mask {
          char c[2];
          int i;
     } bit_mask;
     int i,index,bit_position;
     unsigned char t;
     char xor; /* xor - цвет или наложение */
     char far *ptr = (char far *) 0xB8000000; /* указатель на
                                                 CGA  */
     bit_mask.i = 0xFF3F; /* 11111111 00111111 в двоичном коде */
     /* контроль координат для 4 режима */
     if(x<0 || x>199 || y<0 || y>319) return;
     xor = color_code & 128; /* проверка установки режима xor */
     color_code = color_code & 127; /* маска 7 старших бит */
/* установка bit_mask и color_code в правильное положение */
     bit_position = y%4;
     color_code <<= 2*(3-bit_position);
     bit_mask.i >>= 2*bit_position;
/* поиск соответствующего байта в памяти экрана */
     index = x*40 + (y>>2);
     if(x%2) index+=8152; /* если нечетный, использовать второй
                             банк */
/* запись цвета */
     if(!xor) { /* режим наложения */
          t = *(ptr + index) & bit_mask.c[0];
          *(ptr + index) = t | color_code;
     }
     else { /* режим xor */

           t = *(ptr + index) | (char)0;
          *(ptr + index) = t ^ color_code;
     }
}
/* установка видеорежима */
void mode(mode_code)
int mode_code;
{
     union REGS r;
     r.h.al = mode_code;
     r.h.ah = 0;
     int86(0x10,&r,&r);
}
/* установка курсора в координаты x,y */
void goto_xy(x,y)
int x,y;
{
     union REGS r;
     r.h.ah = 2; /* функция адресации курсора */
     r.h.dl = y; /* горизонтальная координата */
     r.h.dh = x; /* вертикальная координата */
     r.h.bh = 0; /* видеостраница */
     int86(0x10,&r,&r);
}
/* установка цветов диаграмм */
void palette(pnum)
int pnum;
{
```

```
     union REGS r;
     r.h.bh = 1; /* код 4 режима */
     r.h.bl = pnum;
     r.h.ah = 11; /* установка функции цвета */
     int86(0x10,&r,&r);
}
/* сохранение выведенного видеографика */
void save_pic()
{
     char fname[80];
     FILE *fp;
     register int i,j;
     char far *ptr = (char far *) 0xB8000000; /* указатель
          на CGA память */
     char far *temp;
     unsigned char buf[14][80]; /* для размещения содержимого
                                   экрана */
     temp = ptr;

     /* сохранение верхней части текущего экрана */
     for (i=0;i<14;i++)
          for (j=0;j<80;j+=2) {
               buf[i][j] = *temp; /* четный байт */
               buf[i][j+1] = *(temp+8152); /* нечетный байт*/
               *temp = 0;
               *(temp+8152) = 0; /* чистка верхней части
                                    экрана */
               temp++;
          }
     goto_xy(0,0);
     printf(" Имя файла : ");
     gets(fname);
     if(!(fp=fopen(fname,"wb"))) {
          printf(" Невозможно открыть файл \n");
          return;
     }
     temp = ptr;
     /* восстановление верхней части экрана */
     for (i=0;i<14;i++)
          for (j=0;j<80;j+=2) {
               *temp = buf[i][j];
               *(temp+8152) = buf[i][j+1];
               temp++;
          }
     /* сохранение рисунка в файле */
     for (i=0;i<8152;i++) {
          putc(*ptr,fp); /* четный байт */
          putc(*(ptr+8152),fp); /* нечетный байт */
          ptr++;
     }
     fclose(fp);
}
```

ОТОБРАЖЕНИЕ ДИАГРАММ НА ЭКРАНЕ ДИСПЛЕЯ

Если вы сохранили построенную диаграмму в файле, то всегда можете повторно получить изображение данной диаграммы. Для этой цели служит программа SHOW, описанная в данном разделе. Программа выводит диаграмму, находящуюся в файле, имя файла задается в виде аргумента команды. Например, чтобы вывести диаграмму, находящуюся в файле backlog, необходимо ввести команду

```
show backlog
```

Программа show использует функцию load_pic(), предназначенную для изображения диаграмм на экране. (Вы также

можете использовать эту программу для отображения на экране дисплея других графических образов, предварительно созданных и записанных в файл.)

```
/* Простейшая программа восстановления графических образов */
#include "stdio.h"
#include "dos.h"
void load_pic(),mode(),palette(),goto_xy();
main(argc,argv)
int argc;
char *argv[];
{
     if(argc != 2) {
          printf(" Обращение: показать <имя файла>");
          exit(1);
     }
     mode(4);
     palette(0);
     load_pic(argv[1]);
     getch();
     mode(3);
}
/* загрузка графического изображения */
void load_pic(fname)
char *fname;
{
     FILE *fp;
     register int i,j;
     char far *ptr = (char far *) 0xB8000000; /* указатель
          на CGA память */
     char far *temp;
     unsigned char buf[14][80]; /* для размещения содержимого
                                   экрана */
     if(!(fp=fopen(fname,"rb"))) {
          goto_xy(0,0);
          printf(" невозможно открыть файл \n");

          return;
     }
     /* загрузка изображения из файла */
     for (i=0;i<8152;i++) {
          *ptr = getc(fp); /* четный байт */
          *(ptr+8152) = getc(fp); /* нечетный байт */
          ptr++;
     }
     fclose(fp);
}
/* установка видеорежима */
void mode(mode_code)
int mode_code;
{
     union REGS r;
     r.h.al = mode_code;
     r.h.ah = 0;
     int86(0x10,&r,&r);
}
/* установка цветов диаграмм */
void palette(pnum)
int pnum;
{
     union REGS r;
     r.h.bh = 1; /* код 4 режима */
     r.h.bl = pnum;
     r.h.ah = 11; /* установка функции цвета */
     int86(0x10,&r,&r);
```

```
}
 /* установка курсора в координаты x,y */
void goto_xy(x,y)
int x,y;
{
     union REGS r;
     r.h.ah = 2; /* функция адресации курсора */
     r.h.dl = y; /* горизонтальная координата */
     r.h.dh = x; /* вертикальная координата */
     r.h.bh = 0; /* видеостраница */
     int86(0x10,&r,&r);
}
```

НЕКОТОРЫЕ ИНТЕРЕСНЫЕ ИДЕИ ПО МОДИФИКАЦИИ ПРОГРАММ

Вы можете расширить описанные функции, введя параметры, задающие размеры и расположение диаграмм так, чтобы выводить диаграммы разных размеров в различных частях экрана. Например, может быть удобен вывод четырех малых диаграмм, каждая из которых расположена в одном из квадрантов экрана. Вы также можете изменить предложенные функции и работать с графическими режимами более высокой разрешающей способности.