Стандарни функции в С

Входно-изходни операции в езика С.

Езикът Си има няколко функции за въвеждане на данни в програмата (от файл,от клавиатура и др.).Това са следните функции:

clearerr	_fileno	fseek	putc	sprintf
fclose	_flushall	fsetpos	putchar	sscanf
_fcloseall	fopen	_fsopen	puts	_tempnam
_fdopen	fprintf	ftell	_putw	tmpnam
feof	fputc	fwrite	rewind	tmpfile
ferror	_fputchar	getc	_rmtmp	ungetc
fflush	fputs	getchar	scanf	vfprintf
fgetc	fread	gets	setbuf	vprintf
_fgetchar	freopen	_getw	setvbuf	_vsnprintf
fgetpos	fscanf	printf	_snprintf	vsprintf
fgets				

Необходимо е да направим пояснението, че част от тези функции се използват за работа с файлове и ще се разглеждат в следващите упражнения. Тук ще обърнем внимание на по-често използваните функции за вход и изход от към стандартни входно-изходни устройства.

Функция scanf.Предаване на адрес към scanf.Функции gets и getch

Функция scanf

За интерактивен режим на въвеждане на информация най-често се използува библиотечната функция scanf. Тя е функция за вход, еквивалентна на функцията за изход printf. Форматът е:

scanf(<ФорматиращНиз>,<Адрес>,<Адрес>,...)

Много от форматите тип %<буква>, които използува scanf, са същите както за printf (напр. %d, %f, %c). Има обаче една много съществена особеност в работата на scanf - елементите от списъка след форматиращия низ ТРЯБВА ДА БЪДАТ АДРЕСИ! Например

операторът: scanf("%d %d",&a,&b);очаква потребителят да въведе две стойности от цял тип, разделени чрез интервал.

Те ще бъдат присвоени съответно на променливите a и b. Забележете, че се използува операторът & ("Адрес на") за да се предадат адресите на променливите a и b на функцията scanf. Символът, чрез който са разделени форматите във форматиращия низ, не само ги разграничава, но и задава начина, по който потребителят трябва да раздели двете въвеждани стойности. Разделянето чрез интервал показва, че стойностите могат да се разделят чрез произволна комбинация от интервали,табулатори и символи за нов ред. Ако желаете числата да се отделят чрез запетая, поставете във форматиращия низ запетая вместо интервал:

```
scanf("%d,%d",&a,&b);
```

Предаване на адрес към scanf

Когато желаете да въведете символен низ, който да се запази в символен масив, трябва да съобразите, че идентификаторът на масива означава адреса на началото му. Затова в този случай при обръщението към scanf се използува направо името на масива.

Например:

```
main()
{
        char name[30];
        printf("Вашето име ? ");
        scanf("%s",name);
        printf("Здравейте, %s\n",name);
}
```

Тук за съхранение на въведения символен низ се използува масив (**char** name[30]), а не указател към символ (**char** *name). Предпочетен е първият начин,тъй като с него автоматично се отделя място за целия низ, докато при указателя , това място трябва да се задели явно с помощта на специален оператор.

Име **scanf** - осъществява форматиран вход

Употреба
```
int scanf(char *format[,argument...]);
int cscanf(char *format[,argument...]);
int fscanf(FILE *stream, char *format [,argument...]);
int sscanf(char *string,char *format[,argument...]);
int vfscanf(FILE *stream,char *format, va_list argp);
int vscanf(char *format, va_list argp);
int vsscanf(char *string,char *format, va_list argp);
```

Прототип stdio.h

Функциите от фамилията ...scanf обработват входните полета символ по символ и ги превръщат в съответствие със зададен форматиращ низ.

Всички функции ...scanf:

- работят с форматиращ низ (string), който определя начина на интерпретация на входните полета;
- прилагат форматиращия низ върху променлив брой

- входни полета, за да осъществят форматиран вход;
- записват форматирания вход на адрес, даден като аргумент след форматиращия низ (тези адреси се задават или чрез [,argument...] или чрез va_list param.

Когато някоя от функциите регистрира първата си форматна спецификация във форматиращия низ, тя анализира и превръща първото входно поле съгласно тази спецификация, след което запазва резултата в мястото, определено от първия адресен аргумент. Следва анализ, превръщане и записване на останалите входни полета.

Източникът на входа се приема по подразбиране за три от ...scanf функциите:

scanf и **vscanf** четат от стандартния вход stdin;
cscanf чете направо от конзолата.

Останалите четири функции ползуват допълнителен аргумент (първия от списъка на аргументите), който указва източника на входна информация:

fscanf и vfscanf приемат вход от входен поток (определен от stream);
sscanf и vsscanf четат от символен низ от паметта (сочен от string).

Четири от функциите получават списък от адресни аргументи направо при обръщение към тях (**scanf,cscanf, fscanf, sscanf).**

Останалите три (**vscanf, vfscanf, vsscanf**) получават адресните си аргументи от списък от променливи аргументи. Функциите v...scanf са известни като алтернативни входни точки за функциите ...scanf.

За допълнителни сведения относно списък от променливи аргументи вижте дефиницията на va_...

Ето резюме за всяка от функциите:

scanf чете от stdin и съхранява въведеното в мястото, посочено от адресните аргументи &arg1,...,argn.
cscanf чете направо от конзола и съхранява въведеното в мястото, посочено от адресните аргументи &arg1,...,argn.
fscanf чете от посочен входен поток и съхранява въведеното в мястото, посочено от адресните аргументи &arg1,...,argn.
sscanf чете данни, записани като символен низ в паметта и съхранява въведеното в мястото, посочено от адресните аргументи&arg1,...,argn. sscanf не променя символния низ - източник наинформацията.
vscanf работи като scanf, но приема адресни аргументи от масивava_arg на va_list param.
vfscanf работи като fscanf, но приема адресни аргументи от масива va_arg на va_list param.
vsscanf работи като sscanf, но приема адресни аргументи от масива va_arg на va_list param.

ФОРМАТИРАЩ НИЗ

Форматиращият низ присъствува при всяко обръщение към функция scanf и определя как съответната функция ще обработи, преобразува и съхрани входните полета. Адресните аргументи трябва да не са по-малко от включените във форматния низ спецификации. Ако

адресните аргументи са по-малко,последствията са непредвидими и в повечето случаи фатални. Ако адресните параметри са повече от зададените спецификации, излишните се игнорират.

Форматиращият низ е символен низ, съдържащ три типа обекти: символи оставящи интервали (интервал ( ), табулатор (\t) или нов ред (\n)), символи и форматни спецификации.

- ако функция ...scanf регистрира символ за интервали във форматиращия низ, тя чете (но не записва) всички следващи интервални символи от входа до първия различен, без да ги съхранява.
- символите, различни от тези за интервали са останалите ASCII символи (с изключение на символа %). Ако функция ...scanf регистрира такъв символ във форматиращия низ, тя чете, но не съхранява съвпадащите с него символи.
- Форматната спецификация насочва функциите ..scanf да четат символите от входното поле, да ги преобразуват в определен тип стойности и да ги съхраняват в мястото, определено от адресните аргументи.

Завършващи символи за интервали във входните полета не се четат (включително и символ за нов ред), освен ако това не е предвидено явно във форматиращия низ.

Форматни спецификации

Форматните спецификации на ...scanf имат формата:

% [*] [ширина] [F|N] [h|l] символ_за_тип

Винаги се започва със символа "%". Следват полетата в следния ред:

* незадължителен символ, забраняващ присвояването [*]
* незадължителна спецификация за ширина на полето
* [ширина].
* незадължителен спецификатор на размера на указател [F|N].
* незадължителен модификатор на типа на аргумента [F|N|h|l].
* символ за типа [символ_за_тип]

Задължителни компоненти на форматиращия низ

Тук са показани основните характеристики на форматирането на входа, управлявано от незадължителните символи, спецификатори и модификатори:

Символ или спецификация	Обект на управление или спецификатор
*	забранява присвояването на следващото входно поле.
Ширина	минимален брой символи, които да се прочетат; възможно е да бъдат прочетени по-малко символи, ако функцията ...scanf регистрира символ за интервали или символи, които не могат да се преобразуват.
[F|N]	променя подразбиращия се размер на адресните аргументи:
N	Вътрешно сегментен указател - near;
F	Между сегментен указател - far;

[h|l]	тип на аргумента. Променя подразбиращия тип на адресен аргумент (h = указател към тип short **int**; l = указател към тип **long int**).

scanf символи за превръщане на типа

Следващата таблица дава списък на символите, използувани за преобразуване на типа на входа, приеман от всеки от тях и формата, в който ще се съхрани въведената информация.

Информацията в таблицата се основава на приемането, че не е включен никой от незадължителните спецификатори и модификатори. Резултатите от съвместното използуване на тези елементи са показани по-долу.

Символ	Очакван за типа вход	Тип на аргумента
d	Десетично цяло	Указател към **int** (**int** *arg)
D	Десетично цяло	Указател към **long**(**long** *arg)
o	Осмично цяло	Указател към **int** (**int** *arg)
O	Осмично цяло	Указател към **long** (**long** *arg)
i	цяло	Указател към **int** (**int** *arg) (десет., осмично или шестнадесетично.)
I	цяло	Указател към **long**(десет., осмично (**long** *arg) или шестнадесетично.)
u	десетично цяло	Указател към десетично без знак цяло без знак (**unsigned int** *arg)
U	десетично цяло	Указател към десетично без знак цяло без знак (**unsigned long** *arg)
x	шестн. цяло	Указател към **int** (**int** *arg)
X	шестн. цяло	Указател към **long** (**long** *arg)
e	плаваща запетая	Указател към float (float *arg)
E	плаваща запетая	Указател към float (float *arg)
f	плаваща запетая	Указател към float (float *arg)
F	плаваща запетая	Указател към float (float *arg)
c	символ	Указател към символ(**char** *arg)Ако е е дадена и ширина наполето w (напр. %5c), това ще бъде указател към масив от w символа (**char** arg[w]).
s	символен	Указател към масив от символи низ (**char** arg[]).
%	символ %	Не се прави преобразуване.Запазва се символът %.
n	(нищо)	Указател към **int** (**int** *arg). към цяло В този указател се запазва броят на успешно прочетените символи до регистриране на %n.
p	шестнадесетично.	Указател към обект число във(far * или near *) формат %p преобразуването се прави по XXXX:YYYY подразбиране за приетия за или ZZZZ модела на паметта размер на указателите.

Входни полета

Като входни полета се интерпретират:

- всички символи до (но не включително) следващ символ за интервали.
- всички символи до първия, който не може да се преобразува съгласно текущата форматна спецификация.
- символите до n-тия символ, където n е зададената в спецификацията ширина на полето.

Приети конвенции

Тук са изложени основните конвенции, приети и свързани с някои от спецификациите:

%c превръщане Тази спецификация чете следващите символи, включително и символи за интервали. За да се пропуснат символите за интервали и да се прочете следващ, различен от тях символ, използувайте %1s.

%Wc превръщане (W е спецификация за ширина) Адресният аргумент е указател към масив от символи, състоящ се от W елемента (**char** arg[W]).

%s превръщане Адресният аргумент е указател към масив от символи (**char** arg[]).

Размерът на масива трябва да бъде поне (n+1) байта, където n е дължината на символния низ s (в символи). Входното поле завършва с интервал или със символ за нов ред. Нулевият символ се добавя автоматично към символния низ и се запазва като последен елемент в масива.

[филтър_за_търсене] преобразуване

- Символът за тип s може да се замести с поредица от символи, оградени в квадратни скоби. Адресният аргумент е указател към масив от символи (**char** arg[]).
- Квадратните скоби ограждат филтър - набор от символи, които определят символите, които може да включва входното поле.
- Ако първият символ в скобите е "^", то филтърът се инвертира -допускат се всички символи, различни от посочените във филтъра.
- Входното поле е символен низ, без включени в него символи за интервали. Функцията ... scanf чете съответното входно поле до достигане на първия символ, който не се допуска от наложения филтър. Ето два примера:
- %(abcd) търси във входното поле някой от символите a, b, c или d.
- %(^abcd) търси във входното поле символи различни от a, b, c и d.

- %E, %f и %F превръщания за плаваща запетая Числата с плаваща запетая във входните полета трябва да бъдат в следния общ формат:

  [+/-] ddddddddd [.] dddd [E|e][+/-] ddd

  където квадратните скоби означават,че елементът е незадължителен, а с ddd са означени десетични, осмични или шестнадесетични цифри.

- %i, %o, %x, %D, %I, %O, %X, %c, %n превръщания За всяко превръщане, където се допуска указател към символ, **int** или **long**, може да се използува и указател към **unsigned char**, **unsigned int** или **unsigned long**.

Символ за забрана на присвояването (*)

Ако символът "*" следва символа "%" във форматна спецификация, следващото входно поле се анализира, но не се присвоява на следващия адресен аргумент.Полето, чието присвояване е забранено, се очаква да бъде от типа, указан след символа "*".

Спецификаторът [ширина]

Спецификаторът за ширина (n) е десетично цяло, определящо максималния брой символи, който да бъде прочетен от входното поле.

Ако входното поле съдържа по-малко от n на брой символа, функцията ...scanf чете всички символи от него и продължава със следващото поле и следващата форматна спецификация.

Ако преди прочитане на n-символа се срещне символ за интервали (интервал, табулатор, символ за нов ред) или символ, който не може да се преобразува, функцията преминава към следващата спецификация.

Символи, които не могат да се преобразуват са символите, непозволени за зададения формат (напр. 8 и 9 за осмичен формат или I или K за шестнадесетичен или десетичен).

Спецификатор за ширина	Как се променя ширината при съхранение на входа
n	Ще бъдат прочетени, преобразувани и съхранени в текущия адресен аргумент до n на брой символа.

Модификатори за входния размер и за типа на аргумента

Модификаторите за входния размер (F и N) и за типа на аргумента (h или l) определят начина, по който функцията ...scanf интерпретира съответните адресни аргументи (arg).

F и N имат приоритет пред декларирания размер на arg.

h и l показват какъв тип ще се използува за следващите входни данни (h - за short, l – за **long**). Входните данни ще се преобразуват до зададения тип и arg за тях трябва да сочи обект със съответния размер (short за %h, **long** за %l)

Модификатор	Как действува на превръщането (arg)
F	arg се интерпретира като указател тип far
N	Има приоритет пред декларирания размер. arg се интерпретира като указател тип near N не може да се използува при много голям модел на паметта (huge).
h	За d, i, o, u, x типовете: превръща входа в short **int** и го запазва в обект от този тип. За D, I, O, U, X типове - няма ефект. За e, f, c, s, n, p типове - няма ефект.
l	За d, i, o, u, x типовете: превръща входа в **long int** и го запазва в обект от този тип. За e, f типовете: превръща входа в **double** и го запазва в обект от този тип. За D, I, O, U, X типове - няма ефект. За c, s, n, p типове - няма ефект.

а функция ...scanf спира обработката на входа.

Функциите ...scanf могат да прекратят обработката на определено входно поле преди достигане на нормалния му край (символ за интервали) или могат да прекратят обработката на целия вход по много и редица причини.

Функциите ...scanf прекратяват обработката на текущото входно поле и преминават към следващото при следните ситуации:

- Регистриран е символ "*" за забрана на присвояване, следващ символа "%" във форматната спецификация. Съответното входно поле се анализира без да се съхрани;
- При зададена ширина w са прочетени w на брой символа;
- Прочетеният символ не може да се преобразува съгласно текущата форматна спецификация. Например A при десетичен формат;
- Следващият символ от входното поле не е включен в наложения филтър.

Когато функция ...scanf прекрати обработката на входно поле по някоя от тези причини, следващият символ се счита за не прочетен и за начало на следващото входно поле или първия от следваща операция по въвеждане.

Функция ...scanf прекратява обработката на входа при следните ситуации:

- следващият символ от входното поле не отговаря на съответния символ за интервали, зададен във форматиращия низ;
- следващият символ от входен файл е EOF;
- форматиращият низ е изчерпан.

Ако във форматиращия низ се срещне поредица от символи, които не са част от форматна спецификация, те трябва да съвпадат с текущо въвежданите символи от входното поле. Функцията ще ги анализира без да ги запази. Ако се появи символ, различен от тях, той остава във входното поле все едно, че не е четен.

Резултат: Всички функции ...scanf връщат броя на успешно анализираните, конвертирани и съхранени входни полета. Тази стойност не включва полетата, които са само анализирани, без да са съхранени.

Ако някоя от тези функции направи опит да чете след края на файл (или края на символен низ за sscanf и vsscanf), върнатата стойност е EOF.

Ако няма съхранени полета, резултатът е нула.

Ако във форматиращия низ се срещне поредица от символи, които не са част от форматна спецификация,те трябва да съвпадат с текущо въвежданите символи от входното поле.Функцията ще ги анализира без да ги запази. Ако се появи символ, различен от тях, той остава във входното поле все едно, че не е четен.

Преносимост: Функциите scanf, fscanf, sscanf и cscanf са разработени за ОС UNIX.

Функции gets и getchar

Функцията scanf има този недостатък, че интерпретира въведения интервал като знак за край на символния низ. Поради тази причина, в примера за предаване на адрес към scanf е възможно да се въведе само едно име. Проблемът, създаден от наличието на интервал във въвеждания символен низ, може да се преодолее по два начина - като се използуват няколко масива (всяка част, оградена от интервали, се помества в отделен масив) или с помощта на функцията gets. Ето примери:

```
main()
```

```
{
char first[20],middle[20],last[20];
printf("вашето име ? ");
scanf("%s %s %s",first,middle,last);
printf("Здравейте, %s %s %s!\n",first,middle,last);
}
```

В случая, функцията scanf ще чака докато се въведат три символни низа (разделени чрез интервал, табулатор или <**Return**>).
Когато цялата входна информация трябва да се прочете в един символен низ, независимо от съдържащи се вътре интервали и табулатори, обикновено се използува функцията gets.Функцията gets "чете" всички въведени символи, докато срещне символ за край на ред <**Return**>. Самият <**Return**> не се записва в символния низ. Вместо него, в края се поставя нулевият символ (\0). Следващият пример демонстрира приложението на gets:

```
main()
{
char name[60];
printf("Вашето име ? ");
gets(name);
printf("Здравейте, %s\n",name);
}
```

Остана да разгледаме работата на функцията getch. Тя чете само един символ, въведен от клавиатура, без да го отпечатва на екрана. ЗАБЕЛЕЖЕТЕ, че getch не връща прочетения символ като параметър. Самата тя е дефинирана като същинска функция от тип **char** и нейната стойност може да се присвои на променлива от символен тип, да се отпечата и т.н. (напр. ch=getch()).

Функция getchar();

Синтаксис: ch=getshar();
Предназначение: Връща ASCII кода на символ съответстващ на даден натиснат клавиш.

Функция printf

Име **printf** - функция за форматиран изход
Употреба: **int** printf(**char** ***for**mat, ...);
int cprintf(**char** ***for**mat[,argument, ...]);
int fprintf(FILE *stream, **char** ***for**mat [,argument, ...]);
int sprintf(**char** *string, **char** ***for**mat [,argument, ...]);
int vfprintf (FILE *stream, **char** ***for**mat, va_list param);
int vprintf (**char** ***for**mat, va_list param);
int vsprintf (**char** *string, **char** ***for**mat, va_list param);
Прототип stdio.h
Описание Фамилията ...printf осъществява форматиран изход. Всички функции:

- приемат форматиращ низ, който определя начина на форматиране на изхода (това е аргументът **for**mat).

- правилата, зададени чрез форматиращия низ, се прилагат при извеждане на стойностите на променливите, зададени чрез argument... или чрез va_list param.

В три от функциите мястото за изход се определя неявно:

- printf изпраща печата на стандартния изход stdout. Същото прави vprintf.
- cprintf изпраща изхода към конзолата.
- Останалите четири функции имат допълнителен аргумент (първия в списъка), който определя къде ще бъде насочен изходът:
- fprintf и vfprintf насочват изхода към посочения поток.
- sprintf и vsprintf записват изхода в символен низ, разположен в оперативната памет.

При четири от функциите (printf, cprintf, fprintf и sprintf), аргументите се форматират при обръщение към съответната функция.

Останалите три (vprintf, vfprintf и vsprintf) изискват аргументите да са форматирани от променлив списък от аргументи. Функциите v...printf са известни като алтернативни входни точки във функциите ...printf.

Вж. дефиницията на va_... за повече информация.

Ето резюме на ...printf функциите:

printf	изпраща изхода на stdout.
cprintf	отпраща изхода директно на конзолата, без да превръща символа за нов ред в комбинацията CR/LF.
fprintf	изпраща изхода в посочения поток stream.
spirntf	отпраща изхода като нулево прекъснат символен низ в string. Потребителят сам трябва да се грижи да има достатъчно място в string.
vprintf	работи като printf, но взема аргументите от масива va_arg от va_list param.
vfprintf	работи като fprintf, но взема аргументите от масива va_arg от va_list param.
vsprintf	работи като sprintf, но взема аргументите от масива va_arg от va_list param.

За пример за използуване на vprintf вижте va...

ФОРМАТИРАЩИЯТ НИЗ

Форматиращият низ определя как съответната функция ще преобразува, форматира и отпечата аргументите си. Трябва да има достатъчно аргументи за форматиращия низ, в противен случай резултатите са непредвидими и обикновено с тежки последици. Излишните аргументи (за които не е предвидена обработка във форматиращия низ) се игнорират.

Форматиращият низ е символен низ, който съдържа два типа обекти - обикновени символи и превръщащи спецификации: обикновените символи се копират буквално на изхода; превръщащите спецификации вземат поредния аргумент от списъка и го форматират.

Форматни спецификации

Форматните спецификации във функциите ...printf имат следната форма:

% [флагове][ширина][.точност][F|N|h|l] тип

Винаги се започва със символа "%". Следват полетата в следния ред:

- незадължителна последователност от символи-флагове [флагове].
- незадължителна спецификация за ширина на полето [ширина].
- незадължителна спецификация за точност [.точност]
- незадължителен модификатор на размера на аргумента [F|N|h|l].
- символ за типа на превръщане [тип]

Незадължителни компоненти на форматиращия низ

Тук са показани основните характеристики на форматирането на изхода, управлявано от незадължителните символи,спецификатори и модификатори:

Символ или спецификатор	Обект на управление или спецификация
Флагове	подравняване на изхода, в сила за числа, десетични точки, завършващи нули, представки за осмичен и шестнадесетичен запис.
Ширина	минимален брой символи, които да се отпечатат, запълване с интервали и нули
Точност	максимален брой символи, които да се отпечатат; за цели числа - минимален брой цифри, които да се отпечатат.
[F|N|h|l]	променя подразбиращия се размер на аргумента (N = вътрешно сегментен указател - near; F = между сегментен указател - far; h = цяло тип short; l = цяло тип **long**).

Символи за превръщане на типа

Следващата таблица дава списък на символите, използувани за преобразуване на типа, типа на входния аргумент, приеман от всеки от тях и формата на изхода.
Информацията в таблицата се основава на приемането, че не е включен никой от незадължителните спецификатори и модификатори. Резултатите от съвместното използуване на тези елементи са показани по-долу.

Символ	Очакван за типа вход	Тип на аргумента
d	Десетично цяло	Указател към **int** (**int** *arg)
D	Десетично цяло	Указател към **long**(**long** *arg)
o	Осмично цяло	Указател към **int** (**int** *arg)
O	Осмично цяло	Указател към **long** (**long** *arg)
i	цяло	Указател към **int** (**int** *arg) (десет., осмично или шестнадесетично)
I	цяло	Указател към **long**(десет., осмично (**long** *arg) или шестнадесетично.)
u	десетично цяло	Указател към десетично без знак цяло без знак (**unsigned int** *arg)
U	десетично цяло	Указател към десетично без знак цяло без знак (**unsigned long** *arg)
x	шестн. цяло	Указател към **int** (**int** *arg)
X	шестн. цяло	Указател към **long** (**long** *arg)
e	плаваща запетая	Указател към float (float *arg)

E	плаваща запетая	Указател към float (float *arg)
f	плаваща запетая	Указател към float (float *arg)
F	плаваща запетая	Указател към float (float *arg)
c	символ	Указател към символ(**char** *arg)Ако е е дадена и ширина наполето w (напр. %5c), това щебъде указател към масив от wсимвола (**char** arg[w]).
s	символен	Указател към масив от символи низ (**char** arg[]).
%	символ %	Не се прави преобразуване. Запазва се символът %.
n	(нищо)	Указател към **int** (**int** *arg). към цяло В този указател се запазва броят на успешно прочетените символи до регистриране на %n.
p	шестнадесетично.	Указател към обект число във(far * или near *) формат %p преобразуването се прави по XXXX:YYYY подразбиране за приетия за или ZZZZ модела на паметта размер на указателите.

Приети конвенции

Таблицата дава основните конвенции, прилагани при някои от спецификациите:

Символи	Конвенция
e или E	Аргументът се превръща така, че да може да се запише по формата [-]d.ddd...e[+/-]ddd където: • преди десетичната точка стои само една цифра; • броят на цифрите след десетичната точка се определя от точността; • експонентата винаги съдържа три цифри.
f	Аргументът се превръща в десетичен запис във формат [-]ddd.ddd..., където броят на цифрите след десетичната точка е равен на точността (ако е дадена точност, различна от нула).
G или g	Аргументът се печата по e или f формат, когато се използува g. Формат E се използува за G. Формат e се използува само, ако експонентата, получена в резултат от превръщането е: • по-голяма от точността • по-малка от -4.
x или X	При x превръщане за шестнадесетични цифри в изхода се използуват малките букви от латинската азбука (a до f), а при X превръщане - големите (A до F).

Символи-флагове

Флаг	Какво специфицира
-	Ляво подравняване на резултата, запълване с интервали отдясно. Ако флагът не е включен, резултатите се извеждат с дясно подравняване и се запълват отляво с нули или интервали.
+	Превръщане на резултата със знак. Резултатът винаги започва със знак (+ или -).
интер-	Ако стойността е неотрицателна, изходът вал започва с интервал, а не

	с плюс; отрицателните резултати започват с минус.
#	Изисква аргументът да се превърне като се използува "алтернативната форма" (Вж. следващата таблица).

Забележка: флаг "+" има приоритет пред флага "интервал", ако са използувани заедно. Алтернативни форми

Спецификаторът [ширина]

Той задава минималната ширина на полето за извежданата стойност.

Ширината се задава по два начина: пряко, чрез символен низ от десетични цифри и непряко чрез символа "*". Ако се използува "*", следващият аргумент (който трябва да бъде тип **int**) задава минималната ширина на изходното поле.

Недостатъчно голяма стойност или нулева стойност на ширината не предизвикват отрязване на резултата. В тези случаи полето се разширява, за да покаже получения при превръщането резултат.

Спецификатор за ширина	Как се променя ширината на полето за изход
n	Печатат се поне n на брой символа. Ако стойността има по-малко от n символа, тя се допълва с интервали (отдясно, при флаг "-" и отляво в останалите случай).
0n	Печатат се поне n на брой символа. Ако изходната стойност има по-малко от n символа, тя се запълва с нули отляво.
*	Списъкът от аргументи задава спецификатора за ширина. Аргументът, задаващ ширината, предшествува аргумента, който ще бъде форматиран

.

Спецификатор [точност]

Винаги започва с точка (.), за да бъде отделен от ширината. Както и ширината, спецификаторът за точност може да се зададе: пряко, чрез символен низ от десетични цифри и непряко, чрез символа "*". Ако се използува "*", следващият аргумент (който трябва да бъде тип **int**) задава точността.

Ако за ширината или за точността или и за двата спецификатора се използува "*", аргументът, определящ ширината трябва да следва непосредствено спецификаторите, следван от аргумента за точност, след което се поставя аргументът, който ще бъде форматиран.

Спецификатор точност	Как се променя точността за изхода
не е зададен	Точност по подразбиране: • 1 за d, i, o, u, x, X; • 6 за e, E, f;

- всички значещи цифри за g и G;
- до първи символ '\0' за s;
- без ефект за c;

.0 за d, I, o, u, x - точността се приема равна на тази по подразбиране; за e, E, f - не се печата десетична точка.

.n Печатат се n на брой символа или n десетични позиции. Ако изходната стойност има повече от n символа изходът може да бъде отрязан или закръглен (това зависи от избрания формат - Вж. следващата таблица).

* Списъкът от аргументи задава и спецификатора за точност. Той трябва да стои преди стойността за форматиране.

Символ определящ превръщането	Как спецификаторът за точност ".n" действува на превръщането
d i o u x X	.n гарантира, че ще се отпечатат поне n на брой цифри. Ако входният аргумент има по-малко от n цифри, стойността се допълва от ляво с нули, а ако е по-дълъг, той не се отрязва
e E f	.n гарантира, че след десетичната точка ще бъдат отпечатани n на брой символа и че последната цифра ще бъде закръглена.
g G	.n гарантира, че ще бъдат отпечатани поне n на брой значещи цифри.
c	.n не се отразява на изхода.
s	.n гарантира, че ще бъдат отпечатани не повече от n на брой символа.

Модификатор за размера на входния аргумент

Модификаторите за размера на входния аргумент (F, N, h или l) дават размера на следващия входен аргумент:

F - указател тип far
N - указател тип near
h - short int
l - long.

Тези модификатори задават начина, по който ...printf функциите интерпретират типа на съответния входен аргумент (нека го означим с arg).

F и N се прилагат само за аргументи тип указател (%p, %s и %n). Обикновено аргументите за %p, %s и %n са указатели от типа по подразбиране, определен от конкретния

модел на паметта. F налага arg да се интерпретира като между сегментен указател (far), а N като вътрешно сегментен (near).

h и l се прилагат за числени аргументи (цели и реални). Те имат приоритет пред приетия по подразбиране размер на числените данни. l се прилага за данни от цял тип (d, i, o, u, x, X) и за данни от тип плаваща запетая (e, E, f, g, G). h се прилага само за данни от цял тип.

Модификатор:	Как се интерпретира аргументът (arg) за размера?
F arg	се чете като указател тип far
N arg	се чете като указател тип near **N** не може да се използува при много голям модел на паметта (huge).
h arg	се интерпретира като short **int** за d, i, o, u, x или X
l arg	се интерпретира като **long int** за d, i, o, u, x или X. arg се интерпретира като **double** за e, E, f, g, или G.

Резултат: Всяка от функциите връща броя на изведените байтове.

sprintf не включва при броенето нулевия символ. В случай на грешка, функциите връщат EOF.

Преносимост Функциите printf, cprintf, fprintf и sprintf са разработени за ОС UNIX.

vprintf, vfprintf и vsprintf са разработени за ОС UNIX System V.

Пример:

```
#define I 555
#define R 5.5
void main()
{
 int i,j,к,l;
 char buf[7];
 char *prefix = &buf;
 char tp[20];
 printf("Пред-\nставка 6d 6c 8x 10.2e 10.2f\n");
 strcpy(prefix,"%");
 for (i=0;i<2;i++)
 {
  for (j=0;j<2;j++)
   for (к=0;к<2;к++)
    for (l=0;l<2;l++)
    {
     if (i==0) strcat(prefix,"-");
     if (j==0) strcat(prefix,"+");
     if (к==0) strcat(prefix,"#");
     if (l==0) strcat(prefix,"0");
     printf("%5s |",prefix);
     strcpy(tp,prefix);
     strcat(tp,"6d |");
     printf(tp,I);
     strcpy(tp,"");
     strcpy(tp,prefix);
     strcat(tp,"6o |");
     printf(tp,I);
```

```
    strcpy(tp,"");
    strcpy(tp,prefix);
    strcat(tp,"8x |");
    printf(tp,I);
    strcpy(tp,"");
    strcpy(tp,prefix);
    strcat(tp,"10.2e |");
    printf(tp,R);
    strcpy(tp,prefix);
    strcat(tp,"10.2f |");
    printf(tp,R);
    printf(" \n");
    strcpy(prefix,"%");
   }
  }
 }
```

Печат от работата на програмата:

Представка	6d	6c	8x	10.2e	10.2f
%-+#0	+555	01053	0x22b	+5.5e+00	+5.50
%-+#	+555	01053	0x22b	+5.5e+00	+5.50
%-+0	+555	1053	22b	+5.5e+00	+5.50
%-+	+555	1053	22b	+5.5e+00	+5.50
%-#0	555	01053	0x22b	5.5e+00	5.50
%-#	555	01053	0x22b	5.5e+00	5.50
%-0	555	1053	22b	5.5e+00	5.50
%-	555	1053	22b	5.5e+00	5.50
%+#0	+00555	001053	0x00022b	+005.5e+00	+000005.50
%+#	+555	01053	0x22b	+5.5e+00	+5.50
%+0	+00555	001053	0000022b	+005.5e+00	+000005.50
%+	+555	1053	22b	+5.5e+00	+5.50
%#0	000555	001053	0x00022b	0005.5e+00	0000005.50
%#	555	01053	0x22b	5.5e+00	5.50
%0	000555	001053	0000022b	0005.5e+00	0000005.50
%	555	1053	22b	5.5e+00	5.50

Вж. също: ferror fopen fread getc printf puts setbuf

Функции puts и putchar.

Синтаксис: puts(**char** *s)
putchar(**char** c)

Функциите **puts** и **putchar** са предназначени да извеждат на стандартното изходно устройство на символна информация, **putchar** извежда един символ, а **puts** цял стринг.

Математически функции

Програмният език С не е специализиран за извършване на математически операции като някой продукти от рода на MatlLab, MatCad, Matematika и др. За да могат да се

реализират операции от рода на работа с матрици или решаване на диференциални уравнения е необходимо да се напишат съответните функции. Както и да се използват готови математически библиотеки разпространяване от дадени фирми специализирали се в тази област. От друга страна езикът С разполага с множество елементарни математически функции,на базата на кито могат да се създадат разширени библиотеки.

acos	acosl	asin	asinl	atan	atanl
atan2	atan2l	bessel	_cabs	_cabslceil	
ceill	_clear87	_control87	cos	cosl	cosh
coshl	_dieeetomsbin	div	_dmsbintoieee	exp	expl
fabs	fabsl	_fieeetomsbin	floor	floorl	fmod
fmodl	_fmsbintoieee	_fpreset	frexp	frexpl	_hypot
_hypotl	ldexp	ldexpl	ldiv	log	logl
log10	log10l	_lrotl	_lrotr	_matherr	_matherrl
__max	__min	modf	modfl	pow	powl
rand	_rotl	_rotr	sin	sinl	sinh
sinhl	sqrt	sqrtl	srand	_status87	tan
tanl	tanh	tanhl			

В С няма стандартен тип за работа с комплексни числа. Затова в някой компилатори този тип е дефиниран допълнително. При Mictrosoft C/C++ компилаторите е дефиниран като структура в math.h.

```
struct _complex
{
  double x;        // Real component
  double y;        // Imaginary component
}

struct _complexl
{
  long double x;     // Real component
  long double y;     // Imaginary component
};
```

Математически функции

Име **abs** - абсолютна стойност.
Употреба **int** abs(**int** i);
Други **double** cabs (**struct** complex znum);
double fabs (**double** x); **long** labs (**long** n);
Прототип в stdlib.h (abs, labs) math.h (cabs, fabs)
Описание **abs** връща абсолютната стойност на целочисления аргумент i.
Ако обръщението се осъществява, като е включен stdlib.h, abs ще се интерпретира като макродефиниция, която се замества на място в самата програма.Ако не е включен файл stdlib.h (или, ако след като е включен, използувате #undef abs), програмата ще ползува функцията abs, а не макроса.

cabs е макро-дефиниция за изчисляване на абсолютната стойност на комплексното число znum. znum е декларирано като структура от тип complex. Структурата е дефинирана в math.h във вида:

```
struct _complex
        {
                double x,y;
        };
```

Обръщението към **cabs** е еквивалентно на обръщение към sqrt с реалната и комплексната компоненти на znum:

```
sqrt(znum.x*znum.x + znum.y*znum.y)
```

Ако math.h не е включен (или е включен, но е използувано #undef cabs),програмата ще използува функцията cabs вместо макроса.

- fabs изчислява абсолютната стойност на аргумента от тип **double** - z.
- labs изчислява абсолютната стойност на аргумента оттип **long int** - n.

Резултат :

abs връща цяла стойност в интервала от 0 до 32767. Изключение е аргумент -32768, при който функцията дава стойност -32768.

cabs връща абсолютната стойност на znum. Резултатът е от тип **double**. При препълване cabs връща HUGE_VAL и установява за errno:

ERANGE Result out of range Резултат извън допустимия обхват

- Обработката на грешките за cabs може да се модифицира с помощта на функцията matherr.
- fabs връща абсолютната стойност на x, а labs - на n. Не се връща флаг за грешка.

Име	**cabs** - абсолютна стойност на комплексно число
Употреба	**double** cabs(**struct** complex znum);
Прототип в	math.h
Описание	Вж. abs

Име	**ceil** - закръглява
Употреба	**double** ceil(**double** x);
Прототип	в math.h
Описание	Вж. floor

Име	**exp** - експоненциална функция; връща е на степен x;
Употреба	**double** exp(**double** x); **double** frexp(**double** value, **int** *eptr); **double** ldexp(**double** value, **int** expon); **double** log(**double** x);

double log10(**double** x);
double pow(**double** x, **double** y);
double pow10(**int** p);
double sqrt(**double** x);

Прототип в math.h

Описание **exp** изчислява експоненциалната функция е на степен x.
frexp изчислява мантисата x (число тип **double** по- малко от 1) и n (цял тип) така, че value = x . 2.
frexp запазва n в число цял тип, сочено от указателя eptr.
ldexp изчислява стойност тип **double** равна на value по две на степен expon.
log изчислява натурален логаритъм от x.
log10 изчислява десетичен логаритъм от x.
pow изчислява x на степен y.
pow10 изчислява 10 на степен p.
sqrt изчислява +vx.

Резултат: При успешно завършване всички функции връщат изчислените стойности.
exp връща e на степен x
frexp връща мантисата x (<1), така че value = x . 2^1
ldexp връща x, където x = value по 2 на степен expon
log връща ln(x)
log10 връща lg(x)
pow връща p, където p = x на степен y.
pow10 връща x, където x = 10 на степен p.
sqrt връща q, където q = +vx.

Понякога предадените аргументи могат да предизвикат препълване или да имат стойности, с които не може да се осъществи изчислението. При препълване exp и pow връщат стойността HUGE_VAL.
Когато резултатът е недопустимо голям, променливата errno може да получи стойност: **ERANGE**

Резултат извън допустимия обхват errno получава стойност EDOM локална грешка при възникване на някоя от следните ситуации: аргументът x, предаден на log или log10 е отрицателен или равен на нула аргументът x, предаден на pow е отрицателен или равен на нула и аргументът y не е цяло число аргументите x и y, предадени на pow са нули аргументът x, предаден на sqrt е отрицателен.

При възникване на такава грешка: аргументът x, предаден на log или log10 е отрицателен или равен на нула аргументът x, предаден на pow е отрицателен или равен на нула и аргументът y не е цяло число аргументите x и y, предадени на pow са нули аргументът x, предаден на sqrt е отрицателен.

При възникване на такава грешка: log, log10 и pow връщат отрицателната стойност на HUGE_VAL sqrt връща нула.

Обработката на грешките за тези функции може да бъде модифицирана чрез функцията matherr.

Име fabs - абсолютна стойност
Употреба **double** fabs(**double** x);
Прототип в math.h
Описание Вж. abs
Вж. също: farmalloc

Име **rand** - генератор на случайни числа
Употреба **int** rand(**void**);
Други **void** srand(**unsigned** seed);
Прототип в stdlib.h
Описание **rand** използува мултипликативен конгруентен генератор за случайни числа с период 2 на степен 32 и връща псевдослучайни числа в интервала от 0 до 2 на степен 15 минус 1.

Генераторът се реинициализира чрез обръщение към srand с аргумент единица (1). Може да му се зададе нова входна точка, като се извика srand с друга стойност на seed.

Пример:

```
#include  <time.h>
#include  <stdio.h>
#include  <stdlib.h>

main() /*Печата 5 случайни числа в интервала от 0 до 32767 */
 {
  int i; long now;
  srand(time(&now) % 37);  /* започва от случайно място */
  for (i=0;i<5;i++)
    printf("%d\n",rand());
 }
```

Печат от работата на програмата:

```
3809
30742
21197
477
19010
```

Име **srand** - инициализира генератора за случайни числа
Употреба **void** srand(**unsigned** seed);
Прототип в stdlib.h
Описание Вж. rand

Име **trig** - тригонометрични функции
Употреба **double** acos(**double** x);
double asin(**double** x);
double atan(**double** x);
double atan2(**double** y, **double** x);
double cos(**double** x);
double sin(**double** x);
double tan(**double** x);
Прототип math.h
Описание **sin**, **cos** и tan връщат резултата от съответните тригонометрични функции, приложени върху аргумента.

Ъглите се задават в радиани. asin, acos и atan връщат резултата от съответните аркус функции, приложени върху аргумента. Аргументите на asin и acos трябва да бъдат в интервала -1 до 1. Стойности извън този интервал карат asin и acos да върнат нула и errno получава стойност: EDOM Грешка - извън допустимия интервал

atan2 връща аркус тангенс от y/x и дава верни резултати, дори и когато резултантната стойност е близо до pi/2 или -pi/2 (x клони към 0).

sin и cos връщат стойност в интервала от -1 до 1. asin връща стойност в интервала от -pi/2 до pi/2.

acos връща стойност в интервала от 0 до pi. **atan** връща стойност в интервала от -pi/2 до pi/2.

atan2 връща стойност в интервала от -pi до pi.

tan връща стойност за всеки валиден ъгъл. За ъгли близки до pi/2 или -pi/2, функцията връща 0 и errno получава стойност: ERANGE

Резултат извън допустимия обхват. Обработката на грешките за тези функции може да се модифицира с помощта на функцията matherr.

Текстообработка

Програмният език C няма вградени функции за работа с низове и файлове. За тази цел са реализирани множество функции за обработка на низове, отваряне, затваряне, четене и запис в и от текстови файлове. Тези функции са включени в системните библиотеки и са описани в съответните хедар файлове. За да се работи с тях е необходимо само да се декларира хедар файла в потребителската програма. Под стринг в езика C се разбира поредица от символи завършващи с нулев байт (\0). Например: **char** *c="abcdefh"; Разположението на променливите в паметта ще е следното:

a	b	c	d	e	f	h	\0

Под означението '\0' се разбира ASCII символ с код нула. Всяка една функция работеща със стрингове следи за наличието на този символ. При дефинирането на празен стринг в паметта се записва само нулевия байт. Трябва да се съобразяваме с наличието на този байт. Ако искаме да запишем низ с дължина пет символа то трябва да заделим памет за шест символа. Например:

```
char c[6],s[5];
  for(i=0;i<5;i++)
      c[i]= ('a'+char(i));
      c[5]='\0';
  for(i=0;i<5,s[i]=c[i],i++);
```

При това положение "c" ще бъде масив от текстови символи завършващ с нулев байт, т.е ."c" ще бъде правилно определен стринг от 5 символа.За разлика от c s не е стринг,защото в размерите на масива няма нито един нулев символ. При това положение всички функции работещи с низове ще оперират правилно с низа c, а низа s ще има случайна стойност,която зависи от наличието на нулев байт в паметта след областта заета от масива s.

Символни низове

Съгласно K&R, константа символен низ е една символна поредица, която се състои от двойни кавички, текст и отново двойни кавички ("текст"). За да се удължи символна константа на няколко реда трябва да се използува обратна наклонена черта. Си позволява в символна константа да се използуват няколко низа, като след това компилаторът ще ги конкатенира. Ето пример:

```
main()
{
    char *p;
    p = "Ето пример за начина, по който Си"
       "може автоматично \n да съединява дълги"
       "символни низове";
    printf(p);
}
```

Резултатът от работата на програмата показва действието с дълги символни низове.

Microsoft C/C++ съдържа следните функции за опериране с низове.

sscanf	_strdup*	_strnicmp*	strtoк*
_snprintf	strerror	_strnset*	strtol
sprintv	_strerror	strpbrк*	_strupr*
strcat*	strftime	strrchr*	__toascii
strchr*	_stricmp*	_strrev*	tolower
strcmp*	strlen*	_strset*	_tolower
_strcmpi	_strlwr*	strspn*	toupper
strcpy*	strncat*	strstr*	_toupper
strcspn*	strncmp*	_strtime	_vsnprintf
_strdate	strncpy*	strtod	vsprintf

В следващата част ще дадем синтаксиса на част от функциите за работа с низове в Microsoft C/C++

Име **atof** - превръща символен низ в число, записано с плаваща запетая
Употреба **double** atof(**char** *nptr);
Други **int** atoi(**char** *nptr); **long** atol(**char** *nptr);
Прототип в math.h (atof) stdlib.h (atof, atoi, atol)
Описание atof превръща символния низ, сочен от указателя nptr, в число тип **double**.

Функцията разпознава:

- незадължителен низ от символите табулатор и интервал;
- незадължителен знак (плюс/минус)
- следващ ги низ от цифри и евентуално десетична точка;
- незадължително символ "e" или "E", следван (незадължително) от цяло число със знак.

atoi	превръща символния низ, сочен от указателя nptr, в цяло число тип **int**.
atol	превръща символния низ, сочен от указателя nptr, в цяло число

	тип **long**.
atoi и atol	обработват следните поредици: -незадължителен низ от символите табулатор и интервал; -незадължителен знак (плюс/минус) следващ ги низ от цифри.

Трите функции прекъсват превръщането при срещане на първия непозволен символ. Не е предвидена обработка за евентуално препълване.

Резултат Функциите връщат стойността, получена от превръщането на символния низ. Ако низът не може да бъде превърнат в число от съответния тип, функциите връщат стойност нула.

Вж. също: atof

Име **atoi** - превръща символен низ в цяло число
Употреба **int** atoi(**char** *nptr);
Прототип в stdlib.h
Описание: Вж. atof
Вж. също: atof

Име **atol** - превръща символен низ в цяло число тип **long**
Употреба **long** atol(**char** *nptr);
Прототип в stdlib.h
Описание Вж. atof

Име **fcvt** - превръща число с плаваща запетая в символен низ
Употреба **char** *fcvt(**double** value, **int** ndig, **int** *decpt, **int** *sign);
Прототип в stdlib.h

Име stpcpy - копира символен низ в друг символен низ
Употреба **char** *strcpy(**char** *destin, **char** *source);
Прототип string.h
Описание Вж. str...
Вж. също: malloc mem movmem

Име **str**... - фамилия от функции за работа със символни низове
Употреба **char** *strcpy(**char** *destin, **char** *source);
char *strcat(**char** *destin, **char** *source);c
har *strchr(**char** *str, **int** c);
int strcmp(**char** *str1, **char** *str2);
char *strcpy(**char** *dest, **char** *source);
size_t strcspn(**char** *str1, **char** *str2);
char *strdup(**char** *str);
int stricmp(**char** *str1, **char** *str2);
size_t strlen(**char** *str);
char *strlwr(**char** *str);
char *strncat(**char** *destin, **char** *source, size_t maxlen);
int strncmp(**char** *str1, **char** *str2, size_t maxlen);
char *strncpy(**char** *destin, **char** *source, size_t maxlen);

int strnicmp(**char** *str1, **char** *str2, size_t maxlen);
char *strnset(**char** *str, **int** ch, size_t n);
char *strpbrк(**char** *str1, **char** *str2);
char *strrchr(**char** *str,**int** c); **char** *strrev(**char** *str);
char *strset(**char** *str, **int** ch);
size_t strspn(**char** *str1, **char** *str2);
char *strstr(**char** *str1, **char** *str2);
double strtod(**char** *str, **char** **endptr);
long strtol(**char** *str, **char** **endptr, **int** base);
char *strtoк(**char** *str1, **char** *str2);
char *strupr(**char** *str);

Прототип string.h

Описание: Следва описание на функциите str..., наредени по азбучен ред. След всяка функция в скоби е поставена категорията, към която принадлежи.

strcat	добавя един символен низ към друг (конкатенация)
strchr	търси в символен низ първото появяване на определен символ (търсене)
strcmp	сравнява два символни низа (сравнение)
strcpy	копира символен низ в друг (копиране)
strcspn	търси в символен низ първия сегмент, който не включва никакво подмножество на зададен набор от символи (търсене)
strdup	копира символен низ в ново, отделено за него място (копиране)
stricmp	сравнява два символни низа без да зачита разликата главни - малки букви за латинската азбука (сравнение)
strlen	изчислява дължината на символен низ (търсене)
strlwr	анализира символен низ и превръща главните букви от латинската азбука в малки (промяна)
strncat	добавя част от даден символен низ към друг (конкатенация)
strncmp	сравнява част от даден символен низ с част от друг символен низ (сравнение)
strncpy	копира определен брой байтове от един символен низ в друг, като прави скъсяване или добавяне, ако е необходимо (копиране)
strnicmp	сравнява част от един символен низ с част от друг, без да зачита разликата главни - малки букви от латинската азбука (сравнение)
strnset	заменя определен брой символи от символен низ със зададен символ (промяна)
strpbrк	търси в символен низ първото появяване на някой от символите в зададен набор (търсене)
strrchr	търси в символен низ последното появяване на зададен символен низ (търсене)
strrev	реверсира символен низ - записва го отзад напред (промяна)

strset	заменя всички символи в символен низ със зададен символ (промяна)
strspn	търси в символен низ първия сегмент, който се явява подмножество на зададен набор от символи (търсене)
strstr	търси в символен низ появата на зададен символен низ (търсене)
strtod	превръща символен низ в стойност тип **double** (превръщане)
strtoк	търси в символен низ думи, разграничени с разделители, дефинирани в друг символен низ (търсене)
strol	превръща символен низ в стойност тип **long** (превръщане)
strupr	обработва символен низ като превръща малките букви от латинската азбука в главни (промяна).

Разгледаните 25 функции за работа със символни низове изпълняват множество действия, които могат да се разделят в шест категории:

- конкатенация
- промяна
- сравнение
- превръщане
- копиране
- търсене

В следващите разяснения функциите се разглеждат по категории.

Конкатенация

strcat	добавя копие на символния низ source в края на destin. Дължината на получения символен низ е strlen(destin) + strlen(source).
strncat	копира поне maxlen символа от source в края на destin и добавя нулевия символ ('\0'). Максималната дължина на получения символен низ е strlen(destin) + maxlen.

Промяна

strlwr	превръща главните букви (от латинската азбука) от символен низ str в малки. Други промени не настъпват.
strupr	превръща малките букви (от латинската азбука) от символен низ str в главни. Други промени не настъпват.
strset	заменя всички символи от str със символа ch.
strnset	поставя в първите n байта на символния низ str символа ch. Ако n>strlen(str), n се замества с strlen(str).
strrev	реверсира символите (нарежда ги отзад напред) от символен низ (без завършващия нулев символ).

Сравнение

strcmp	сравнява str1 със str2
stricmp	сравнява str1 със str2 без да зачита разликата главни - малки букви от латинската азбука.
strncmp	прави същото сравнение както strcmp, но преглежда само maxlen на брой символа.
strnicmp	сравнява максимум maxlen на брой байта от str1 със str2, без да зачита разликата главни - малки букви от латинската азбука.

Всички функции за сравнение връщат стойност (<0, 0 или >0) в зависимост от резултата от сравнението на str1 (или част от него) с str2 (или част от него).

Превръщане

strtod превръща символен низ str в стойност тип **double**. str е последователност от символи, които могат да се интерпретират като част от стойност тип **double**. Трябва да е спазен следния общ формат:

[интрв][знак][ddd][.][ddd][e/E[знак]ddd]

където [интрв] означава незадължителни интервали, [знак] е знак плюс или минус, а с ddd за означени цифри.

Ето примери:

+1231.1082 e-1
-2000.999

Функцията strtod прекратява четенето на символния низ след първия невалиден за преобразуване символ. Ако endptr не е NULL, функцията назначава указателя да сочи символа, в който е прекъсната обработката (*endptr=&stopper).

strtol превръща символния низ str в стойност тип **long**. str е последователност от символи, които могат да бъдат интерпретирани като такива от стойност тип **long**. Общият формат е:

[интрв][знак][0][x][ddd]

където [интрв] означава незадължителни интервали, [знак] е знак плюс или минус, [0] и [x] са незадължителни (за означаване на осмични или шестнадесетични числа), а с ddd за означени цифри.

Функцията strtol прекратява четенето на символния низ след първия невалиден за преобразуване символ.

Ако base е между 2 и 36, цялото тип **long** се представя при зададената база за бройна система.

Ако base е 0, първите символи от str определят базата, както следва:

Първи символ	Втори символ	str се интерпретира като:
0	1 - 7	осмично
0	x или X	шестнадесетично
1 - 9	----	десетично

Следните стойности на base се интепретират като невалидни:
base < 0
base = 1
base > 36

Невалидна стойност за base води до резултат 0 и указател endptr, сочещ началото на символния низ.

Ако str се интерпретира като осмична стойност, символи извън интервала 0 до 7 се смятат за невалидни.

Ако str се интерпретира като десетична стойност, символи извън интервала 0 до 9 се смятат за невалидни.

Аналогично се обработват стойности за останалите бази.

Копиране

strcpy копира символния низ source в destin. В края се прибавя нулевият символ.

strncpy копира точно maxlen на брой символа от source в destin, като скъсява или удължава destin. destin може да не бъде нулево прекъснат, ако дължината на source е maxlen или повече от maxlen.

stpcpy копира байтовете от source в destin и свършва след прекопиране на нулевия символ от source. stpcpy(a,b) е същото както strcpy(a,b), но върнатите стойности се различават.

strcpy strcpy(a,b) връща a, докато stpcpy(a,b) връща a+strlen(b).

strdup прави копие на str, като отделя за него място чрез неявно обръщение към malloc. Отделеното място е (strlen(str)+1) байта.

Търсене

strchr анализира символния низ str в права посока, като търси първото появяване на символа ch в него. Нулевият символ се счита за част от символния низ, така че:

strchr(strs,0)

връща указател към завършващия нулев символ на strs.

strrchr анализира символен низ от края към началото и търси появяването на символа ch. Така се намира последното му появяване в низа. Нулевият символ се счита за част от низа.

strpbrк търси в символния низ str1 първото появяване на някой от символите, включени в str2.

strspn връща дължината на началния сегмент от str1, който се състои изцяло от символи, включени в str2.

strcspn връща дължината на началния сегмент от str1, който се състои изцяло от символи, които не са включени в str2.

strstr търси в str2 първото появяване на подмножеството str1.

strtoк разглежда str1 като съставен от поредица от нула или повече думи, разделени чрез зони от един или повече символи- разделители. Разделителите са включени в str2.

Първото обръщение към strtoк връща указател към първия символ на първата намерена дума в str1 и записва в str1 нулевия символ, непосредствено след намерената дума. Следващи обръщения с NULL в първия аргумент ще продължат през str1 по същия начин до момента, в който няма повече думи.

Символният низ str2, съдържащ разделителите, може да бъде различен при всяко обръщение. При изчерпани думи от str1, функцията връща указател със стойност NULL.

Резултат Следват изброени резултатите връщани от различните функции:

stpcpy връща destin +strlen(source)

strchr връща указател към първото появяване на ch в str. Ако ch не е намерен, върнатата стойност е NULL. strcmp, stricmp, strncmp и strnicmp връщат стойност:

<0 ако str1 < str2
=0 ако str1 = str2
>0 ако str1 > str2

При сравнение, функциите зачитат знака.

strcpy връща destin

strdup връща указател към мястото от паметта, където е създаден дублиращият символен низ. Ако не е отделено място, указателят е със стойност NULL.

strlen връща броя символи, включени в str, без да се смята нулевият символ.

strncpy връща destin

strpbrк връща указател към първото появяване на който и да е от символите в str2. Ако няма такъв (никой от символите в str2 не се среща в str1), върнатата стойност е NULL.

strrchr връща указател към последното появяване на символа ch. Ако ch не е намерен в str, функцията връща NULL.

strrev връща указател към реверсирания символен низ. Не се връща грешка.

strstr връща указател към елемента от str2, съдържащ str1 (сочи мястото на str1 в str2). Ако str1 не се среща в str2, върнатата стойност е NULL.

Пример:

```
/* strtoк - Примерът демонстрира използването на strtoк за разбора на дата.
Забележете, че за да се прави разбор на дати в различен формат, трябва
да се зададе символен низ с разделители. Забележете, че разделителите
не се включват в резултата */

#include <stdio.h>
#include <string.h>
void main()
 {
  char *ptr;
```

```
ptr = strtok ("Февруари.14,1989",".,-/");
}
```

Различен формат, трябва да се зададе символен низ с разделители. Забележете, че разделителите не се включват в резултата

```
#include  <stdio.h>
#include  <string.h>
void main()
 {
  char *ptr;
  ptr = strtok ("Февруари.14,1989",".,-/");
  printf ("ptr = %s\n",ptr);
  ptr = strtok (NULL,".,-/");
  printf ("ptr = %s\n",ptr);
 }
```

Печат от работата на програмата:

```
ptr = Февруари
ptr = 14
```

Вж. също: perror

Име **toascii** - превръща целочислена стойност в същински ASCII формат, като зачита само младшите 7 бита.

Прототип ctype.h

Употреба

```
int toascii(int c)
#define ((c) & 0x7f)
int tolower(int c);
int toupper(int c);
int _tolower(int c);
int _toupper(int c);
#define _toupper(c) ((c) + 'A' - 'a')
#define _tolower(c) ((c) + 'a' - 'A')
```

Описание

toascii функция, която превръща цялото с в същински ASCII формат чрез изчистване на всичко без младшите седем бита, така че получената стойност е в интервала от 0 до 127. Предназначена е за осигуряване на съвместимост с други изчислителни системи.

tolower функция, която превръща цялата стойност с (в интервала от EOF до 255) в съответната й "малка" стойност (ако стойността е "главна"). Действието на функцията може да се опише така:

(c) + 'A' - 'a'

toupper интервала от EOF до 255) в съответната й "главна" стойност (ако стойността е "малка"). Действието на функцията може да се опише така:

(c) + 'a' - 'A'

_tolower макродефиниция, която работи както tolower, но трябва да се използува, само ако е сигурно, че с представя "главна" буква. стойност (ако стойността е "малка"). Действието на функцията може да се опише така:

(c) + 'a' - 'A'

_tolower макродефиниция, която работи както tolower, но трябва да се използува, само ако е сигурно, че с представя "главна" буква.

_toupper макродефиниция, която работи както toupper, но трябва да се използува,

само ако е сигурно, че с представя "малка" буква. За използуването на _tolower и _toupper трябва да се включи заглавният файл ctype.h.

Резултат При успешна работа всяка от функциите връща превърната стойност. При грешка не се връща резултат.

Вж. също: toascii

Име **_tolower** - превръща "главни" символи в малки
Употреба **#include** <ctype.h>
int _tolower(**int** c);
Прототип в ctype.h
Описание Вж. toascii

Име **_toupper** - превръща "малки" символи в главни
Употреба **#include** <ctype.h>
int _toupper(**int** c);
Прототип в ctype.h
Описание Вж. toascii
Вж. също: toascii

Име **sscanf** форматиран вход от символен низ.
Синтаксис: **int** sscanf(**char** *buffer,**char** ***for**mat[,argument]...);
Прототип <stdio.h>
Виж също: scanf, sprintf, fscanf, _cscanf
Синтаксис: **int** sprintf(**char** *buffer,**char** ***for**mat[,argument]...);
int _snprintf(**char** *buffer,size_t count, **char** ***for**mat [, argument]...)
Прототип в <stdio.h>
Резултат: записва форматирани данни в <buffer>. Функцията връща размера на записания низ в байтове. За _snprintf, променливата <count> задава максималния размер на стринга. Ако размерът на сртинга е по-голям се връща -1. Форматиращите параметри са същите като при функцията printf.

Виж също: sscanf, printf, _vsnprintf, vsprintf, fprintf,_cprintf

Прототип: <string.h>

Синтаксис: size_t strcspn(**char** *string1, **char** *string2);
size_t _fstrcspn(**char** __far *string1, **char** __far *string2);
Резултат: Връща дължината на началния сегмент,който изцяло не се състои от елементи съдържащи се в string2.
Виж също: strspn, strpbrк, strtoк, strstr, strncmp, strchr

Прототип: <time.h>
Синтаксис: **char** *_strdate(**char** *datestr);
char *_strtime(**char** *timestr);
Резултат: Връщат форматирани стрингове съдържащи датата или времето.
Виж също: asctime, ctime, time

Прототип: <string.h>
Синтаксис: **char** *_strdup(**char** *string);
char __far *_fstrdup(**char** __far *string);
char __near *_nstrdup(**char** __far *string);
Резултат: указател при успех или NULL при неуспех
Виж също: strcpy, strcat, strncpy, _strset

Прототип: <string.h>
Синтаксис: **char** *strerror(**int** errnum);
char *_strerror(**char** *string);
Резултат: Връща указател към стринг съдържащ съобщение за възникнала грешка.
Виж също: perror, clearerr, errno, ferror, _sys_nerr

```
/* CASE.C демонстрира действието на долу изброените функции с низове и символи.
* Илюстрирани функции:
*     _strupr        toupper        _toupper
*     _strlwr        tolower        _tolower
*     _strrev        __toascii
*/

#include  <string.h>
#include  <stdio.h>
#include  <ctype.h>

char mstring[] = "Dog Saw Dad Live On";
char *ustring, *tstring, *estring;
char *p;

void main()
{
 printf( "Original:\t%s\n", mstring );
 /* Големи и малки букви */
 ustring = _strupr( _strdup( mstring ) );
 printf( "Uppercase:\t%s\n", ustring );
 printf( "Lowercase:\t%s\n", _strlwr( us tring ) );
/* Реверс на всички символи. */
 tstring = _strdup( mstring );
 for( p = tstring; *p; p++ )
 {
   if( isupper( *p ) )   *p = tolower( *p );
   else  *p = toupper( *p );

/* This alternate code (commented out) shows how to use _tolower
    * and _toupper for the same purpose.
   if( isupper( *p ) )   *p = _tolower( *p );
   else if( islower( *p ) )   *p = _toupper( *p );     */
 }
 printf( "Toggle case:\t%s\n", tstring );
 estring = _strdup( mstring );
 for( p = estring; *p; p++ )
   *p = *p | 0x80;
 printf( "Encoded:\t%s\n", estring );
```

```
 for( p = estring; *p; p++ )
    *p = __toascii( *p );
 printf( "Decoded:\t%s\n", estring );
 printf( "Reversed:\t%s\n", _strrev( ustring ) );
}


/* CMPSTR.C илюстрира функциите за работа със стрингове и блокове памет:
 *      memcmp      _memicmp    strncmp      _strnicmp
 *      strcmp      _stricmp        _strcmpi
 */

#include  <memory.h>
#include  <string.h>
#include  <stdio.h>
char string1[] = "The quick brown dog jumps over the lazy fox";
char string2[] = "The QUICK brown fox jumps over the lazy dog";
void main()
{
 char tmp[20];
 int result;
 printf("Сравняване на низове:\n\t\t%s\n\t\t%s\n\n", string1,string2 );
 printf( "функция:\tmemcmp\n" );
 result = _memcmp( string1, string2 , 42 );
 if( result > 0 )    strcpy( tmp, "по-голям отколкото" );
 else if( result < 0 )    strcpy( tmp, "по-малък отколкото" );
 else    strcpy( tmp, "равен на" );
 printf( "Резултат:\t\tСтринг1 е %s Стринг2\n\n", tmp );
 printf( "Функция:\t_memicmp\n" );
 result = _memicmp( string1, string2, 42 );
 if( result > 0 )    strcpy( tmp, "по-голям от " );
 else if( result < 0 )   strcpy( tmp, "по-малък от" );
 else    strcpy( tmp, "равен на" );
 printf( "Резултат\t\tСтринг 1 is %s Стринг 2\n\n", tmp );
 printf( "Функция:\tstrncmp\n" );
 result = strncmp( string1, string2 , 42 );
 if( result > 0 )    strcpy( tmp, "по-голям от");
 else if( result < 0 )    strcpy( tmp, "равен на");
 else    strcpy( tmp, "по-голям от);
 printf( "Result:\t\tString 1 is %s string 2\n\n", tmp );
 printf( "Функция:\_tstrnicmp\n" );
 result = _strnicmp( string1, string2, 42 );
 if( result > 0 )    strcpy( tmp, "по-голям от" );
 else if( result < 0 )    strcpy( tmp, "по-малък от" );
 else    strcpy( tmp, "равен на" );
 printf( "Result:\t\tString 1 is %s string 2\n\n", tmp );
 printf( "Функция:\tstrcmp\n" );
 result = strcmp( string1, string2 );
 if( result > 0 )    strcpy( tmp, "по-голям от" );
 else if( result < 0 )    strcpy( tmp, "по-малък от" );
 else    strcpy( tmp, "равен на" );
 printf( "Result:\t\tString 1 is %s string 2\n\n", tmp );
 printf( "Функция:\t_stricmp or _strcmpi\n" );
 result = _stricmp( string1, string2 );
if( result > 0 )    strcpy( tmp, "по-голям от" );
 else if( result < 0 )    strcpy( tmp, "по-малък от" );
 else    strcpy( tmp, "равен на" );
 printf( "Result:\t\tString 1 is %s string 2\n", tmp );
}
```

```
/*  Отделя думите от даден файл */
#include <stdio.h>
int main( int argc, char *argv[], char *envp[])
{
     int br,k;
     char Buff[1000];
     FILE *fp;
     if ( argc < 2 )  {  printf("\nSyntax : %s   File_Name",argv[0]);  return 1; }
     if((fp=fopen(argv[1],"rb"))==NULL) {   printf("\nError open file %s",argv[1]);  return 2;   }
    br = fread(Buff,1,1000,fp);
    for ( k =0;k< br; k++)
    {
      switch(Buff[k])
      {
        case '.':
        case ',':
        case ':':
        case '"':
        case '\\':
        case '-':
        case '?':
        case '!':
        case '*':
        case ';':
        case ')':   printf("\n%c\n",Buff[k]);     break;
        case ' ' : printf("\n"); break;
        default : printf("%c",Buff[k]);
      }
    }
  fclose(fp);
}

// Преобразува всички символи от даден файл в от малки в големи
#include <stdio.h>
#include <string.h>
#include <malloc.h>
FILE *fp1,*fp2;
int main(int v,char *c[])
{
   char s[256];
   char *s1=(char*)malloc(256);
   if (v<3) { printf("\n Няма достатъчно параметри"); exit(1);}
   if ((fp1=fopen(c[1],"rt"))==NULL)  {printf("\n неможе да се отвори файлът %s",c[1]); free(s1);exit(2);}
   if   ((fp2=fopen(c[2],"w"))==NULL){printf("\n     немеже  да  се  отвори  файлът  %s",c[1]);fclose(fp1);
free(s1);exit(3);}
    while(! feof(fp1))
    {
     fgets(s,255,fp1);
     s1 = _strupr(s);
     fputs(s1,fp2);
    }
    fclose(fp1);fclose(fp2);free(s1);
    return 0;
}
```

Основна структура на файловата система

Тома е основата на файловата система на DOS. MS-DOS се обръща към дисковите устройства като към том.Всеки том, има своя собствена структура, без значение дали е

предназначен за дискета или твърд диск. Размерът на диска не е важен за структурата, защото той засяга само броя на отделните структури от данни.

Етикет на том

Въпреки, че не е задължително всеки том се свързва с етикет. Всеки том има своя собствена ROOT директория, която може да има поддиректории. Достъпът до отделните директории и поддиректории се осъществява посредством набор от функции от прекъсване 21H или функциите от библиотеките на C.

Сектори

Dos подразделя всеки том на серии от сектори, организирани последователно. Всеки сектор съдържа определен брой байтове (обикновено 512) и е свързан с номер на логически сектор, започвайки от 0. Един 10M том съдържа 20480 сектора, организирани в логически сектори с номера от 0 до 20479.Dos неможе да контролира физическото разположение на секторите.Това се контролира от драйвера на устройството. При достъп до файлове Dos превръща обръщенията към файловете към достъп до конкретните логически сектори. Това става посредством структура от данни наречена FAT (таблица за разположението на файловете).

Структурата на едно устройство за съхраняване на данни е следната:

BOOT сектор - Съдържа името на производителя, таблици за формата на дисковото устройство, програма за първоначално инициализиране на системата

FAT - Съдържа разположението на файловете по клъстери на диска.

Копие на FAT

ROOT - Главна директория на тома. Съдържа наименованието на тома, наименование на файлове и поддиректории, размер и първи клъстер на файловете.

Забележка: Размерът на един клъстер е различен за различните типове носители.

BOOT сектор

BOOT сектора съдържа цялата необходима информация за достъпа до различните области и структури от данни. Dos изгражда тази структура по време на форматирането на носителя. BOOT сектора е с еднаква структура за всичките типове носители и е разположен на логически сектор 0. Структурата на BOOT е следната:

Структура на BOOT сектора

Адрес	Съдържание	Размер
00H	Преход към BOOT програма (E9xxx или EBxx90)	3 байта
03H	Име на производителя и номер на версията	8 байта
0BH	Байтове на сектор	1 дума
ODH	Сектори на клъстер	1 байт

0EH	Брой запазени сектори	1 дума
10H	Брой FAT таблици	1 байт
11H	Брой елементи в ROOT директорията	1 дума
13H	Брой сектори в том	1 дума
15H	Описание на носителя	1 байт
16H	Брой сектори във FAT	1 дума
18H	Сектори на пътечка	1 дума
1AH	Брой на четящо/записващи глави	1 дума
1CH	Брой скрити сектори	1 дума
1EH-1FFH	BOOT програма	

След като се включи компютъра управлението се поема от BIOS. Той зарежда физически сектор 0 от диска или дискетата в паметта. Проверява физически сектор 0 за информация. Зарежда информацията от boot след което предава управлението на адрес 0 от заредения сектор. На този адрес се намира инструкция JMP към истинското начало на стартиращата (BOOT) програма. Тази програма bootstrap манипулира зареждането и стартирането на Dos чрез индивидуални системни параметри. BOOT сектора може да се следва от няколко запазени сектора и могат да съдържат допълнителен bootstrap код. Броя на тези сектори е записан в BPB полето започващо от отместване 0EH; 1 в това поле показва че boot сектора не е следван от допълнителни сектори.

В интервала от 0BH до 1EH се намира таблица описваща физическият формат на диска. Тази таблица е част от BPB. Информацията в BPB се използва от BIOS сервизните функции извиквани чрез прекъсване 13H.

Таблица за разположението на файловете (FAT)

Dos трябва да знае кои сектори са свободни преди да добави нов файл на устройството. Тази информация се съдържа в стуктура от данни наречена FAT и разположена веднага след запазве на област за bootstrap. Всеки елемент на FAT отговаря на един или няколко съседни сектора отговарят на един клъстер. Позиция 0DH от BOOT сектора показва броят на секторите на клъстер, като част от BPB. Допустими стойности са само степени на 2(1,2,4,8, и т.н.). На XT твърдите дискове един клъстер има размер от 8 сектора , а при AT размерът на клъстера е 4 сектора. Броят на секторите обхващащи един клъстер зависи от размера на носителя.

Сектори на клъстер

Устройство	Сектор на клъстер
Едностранно дисково у-во	1
Двустранно дисково у-во	2
AT твърд диск	4
XT твърд диск	8

FORMAT командата не се ограничава до тези данни.

Вход/Изход и работа с файлове

Файловете в езика Си обикновено се третират като поредица от байтове, а интерпретирането им като текст или данни е до голяма степен проблем на програмиста

(въпреки че може да се използува функцията fopen с идентификатори t - за текс- тов формат и b- за двоичен).

Функции за работа с файлове

Функции извършващи операции над файлове и директории:

_access	_fullpath	_makepath	_searchenv
_chdir	_getcwd	_mkdir	_setmode
_chdrive	_getdcwd	_mktemp	_splitpath
_chmod	_getdrive	remove	_stat
_chsize	_isatty	rename	_umask
_filelength	_locking	_rmdir	_unlink
_fstat			

Функции осъществяващи контрол над входа и изхода от файлове на ниско ниво :

_close	_dup2	_read	_write
_creat	_eof	_sopen	
_commit	_lseek	_tell	
_dup	_open	_umask	

Системни функции осъществяващи управление над файловата система:

_bios_disk	_dos_creat	_bios_equiplist	_dos_creatnew
_dos_open	_dosexterr	_dos_read	_dos_findfirst
_dos_findnext	_dos_setfileattr	_dos_getdiskfree	_dos_setftime
_dos_getdrive	_dos_getfileattr	_dos_getftime	_dos_write
_dos_close			

Функции осъществяващи контрол над входа и изхода от файлове на високо ниво :

clearerr	_fileno	fseek	putc	fclose
_flushall	fsetpos	_fcloseall	fopen	_fsopen
puts	_tempnam	_fdopen	fprintf	ftell
_putw	tmpnam	feof	fputc	fwrite
rewind	tmpfile	ferror	_fput**char**	getc
_rmtmp	ungetc	fflush	fputs	vfprintf
fgetc	fread	setbuf	_fget**char**	freopen
setvbuf	_vsnprintf	fgetpos	fscanf	_snprintf
vsprintf	fgets			

Име **access** - определя достъпността на файл
Употреба **int** access (const **char** *filename, **int** amode);

Прототип io.h

Описание Проверява дали файл с даденото име съществува и дали може да бъде четен, да се записва на него или да се изпълнява. filename сочи символния низ, съдържащ името на файла.

Поредицата от битове в amode се конструира по следния начин:

06 Проверка за достъп за четене и запис
04 Проверка за достъп за четене
02 Проверка за достъп за запис
01 Дали файлът може да се изпълни
00 Проверка за съществуване на файла.

Забележка: В ДОС всички съществуващи файлове са с позволен достъп за четене (amode=04), така че 00 и 04 дават един и същи резултат. По същата причина 02 и 06 са еквивалентни.

Ако filename е директория, access определя дали тя съществува или не.

Резултат Ако резултатът от заявената проверка е положителен (заявеният достъп е разрешен), функцията връща стойност 0. В противен случай върнатата стойност е -1 и errno заема някоя от стойностите:

ENOENT Ненамерен файл или път
EACCES Достъпът отказан

Преносимост Разработена за ОС UNIX.

Пример:

```
include <stdio.h>
include <io.h>
/* Връща 1 ако файлът съществува и 0 в противен случай */

int file_exists(char *filename)
{
   return (access(filename,0)==0);
}

void main()
{
  printf("Съществува ли файл NOTE.TXT: %s\n",
   file_exists("NOTE.TXT") ? "Да" : "Не");
}
```

Резултат от работата на програмата:
Съществува ли файл NOTE.TXT: Не

Вж. също: trig

Име bios_disk - В/И операции с дискови устройства

Синтаксис: **unsigned** _bios_disk(**unsigned** service,**struct** _diskinfo_t *diskinfo);
service: _DISK_RESET, _DISK_STATUS, _DISK_READ, _DISK_WRITE, _DISK_VER**IF**Y, _DISK_**FOR**MAT

Описание Функцията използува прекъсване 0x13 от BIOS, за да осъществи

операции с диск.
Прототип bios.h
Резултат: стойност в регистъра AX

service дефинира вида на операцията, която да се извърши. Стойността на cmd определя дали останалите параметри са необходими. За микрокомпютри IBM PC, XTили AT cmd може да заема следните стойности:

_DISK_RESET Привеждане дискетната система в изходно състояние(контролерът осъществява хардуерно рестартиране на устройството). Останалите параметри се игнорират.

_DISK_STATUS Връща състоянието, регистрирано при последната операция с диск. Останалите параметри се игнорират.

_DISK_READ Чете един или повече сектори от диска в оперативната памет. Секторът, от който да започне четенето, се определя чрез параметрите head, track и sector, а nsect задава броя на секторите, които да бъдат четени. Данните се четат на сектори от по 512 байта и се прехвърлят в buffer.

_DISK_WRITE Записва един или повече сектора от оперативната памет на диска. Секторът, от който да започне записът, се определя чрез параметрите head, track и sector, а nsect задава броя на секторите, които да бъдат записани. Данните се четат от buffer и се записват по 512 байта в сектор.

_DISK_VER**IF**Y Проверява един или повече сектора. Секторът, от който да започне проверката, се определя чрез параметрите head, track и sector, а nsect задава броя на секторите за проверка.

DISK**FOR**MAT Форматира пътечка от диска, която се задава чрез head и track. buffer сочи към таблица от заглавни блокове за сектори, които да се запишат на пътечката. Подробни сведения за вида на таблицата и формата на операциите са дадени в справочната техническа литература за MS-DOS.

```
struct _diskinfo_t
{
  unsigned drive;    // Drive: 0-0x7f floppy, 0x80=0xff fixed disk
  unsigned head;     // Номер на файловият манипулатор
  unsigned track;    // Номер на пътечка
  unsigned sector;   // Номер на стартов сектор
  unsigned nsectors; // Брой сектори за четене, запис или сравняване
  void __far *buffer;// Буфер използван при чеетене, запис или сравнение
};
```

drive е число, което задава номера на дисковото устройство, което да бъде използувано: 0 - за първо флопи-дисково устройство, 1 - за второ флопи-дисково устройство, 2 - за трето и т.н. За твърди дискове стойност на drive 0x80 се отнася за първи твърд диск, 0x81 - за втори и т.н.

При твърд диск, drive задава физическото дисково устройство, а не раздел (логически обособена част от физическия диск). Ако дискът е разделен логически на раздели (partitions),

приложната програма трябва сама да се грижи за интерпретиране на таблицата, съдържаща информация за тях.

Резултат Функцията връща байт на състоянието съставен от битове със следното значение:

0x00	Успешно завършване на операцията
0x01	Неправилна команда
0x02	Ненамерен етикет на адреса
0x04	Ненамерен запис
0x05	Неуспешно рестартиране на устройството
0x07	Drive parameter activity failed
0x09	Опит за достъп до паметта с пряк достъп с преминаване 64К-граница. (Attempt to DMA across 64K boundary)
0x0B	Регистриран флаг за повредена пътечка (Bad track flag detected)
0x10	Bad ECC on disk read
0x01	Неправилна команда
0x02	Ненамерен етикет на адреса
0x04	Ненамерен запис
0x05	Неуспешно рестартиране на устройството
0x07	Drive parameter activity failed
0x09	Опит за достъп до паметта с пряк достъп с преминаване 64К-граница. (Attempt to DMA across 64K boundary)
0x0B	Регистриран флаг за повредена пътечка (Bad track flag detected)
0x10	Bad ECC on disk read
0x11	ECC corrected data error
0x20	Грешка в контролера (Controler has failed)
0x40	Неуспешно позициониране на главата (Seek operation failed)
0xBB	Недефинирана грешка (Undefined error occurred)
0xFF	Sense operation failed

Преносимост Само за микрокомпютри IBM PC и съвместими

```
/* Тази програма илюстрира действието на следните функции:
 *     _bios_disk          _dos_getdiskfree
 */

include <stdio.h>
include <conio.h>
include <bios.h>
include <dos.h>
include <stdlib.h>

char __far diskbuf[512];

void main( int argc, char *argv[] )
{
   unsigned status = 0, i;
   struct _diskinfo_t di;
   struct _diskfree_t df;
   unsigned char __far *p, linebuf[17];

   if( argc != 5 )
```

```
    {
      printf( "  SYNTAX: DISK <driveletter> <head> <track> <sector>" );
      exit( 1 );
    }

    if( (di.drive = toupper( argv[1][0] ) - 'A' ) > 1 )
    {
      printf( "Must be floppy drive" );
      exit( 1 );
    }
    di.head    = atoi( argv[2] );
    di.track   = atoi( argv[3] );
    di.sector  = atoi( argv[4] );
    di.nsectors = 1;
    di.buffer  = diskbuf;

    /* Чете информация за размера на диска */
    if( _dos_getdiskfree( di.drive + 1, &df ) )
      exit( 1 );

    /* Чете до три пъти от диска.  */
    for( i = 0; i < 3; i++ )
    {
      status = _bios_disk( _DISK_READ, &di ) >> 8;
      if( !status )
        break;
    }

    /* Извежда един сектор. */
    if( status )
      printf( "Error: 0x%.2x\n", status );
    else
    {
     for(p=diskbuf,i=0;p<(diskbuf + df.bytes_per_sector);p++)
      {
        linebuf[i++] = (*p > 32) ? *p : '.';
        printf( "%.2x ", *p );
        if( i == 16 )
        {
            linebuf[i] = '\0';
            printf( " %16s\n", linebuf );
            i = 0;
        }
      }
    }
    exit( 1 );
  }
```

Име	**chdir** - променя текущо активната директория
Употреба	**int** chdir(**char** *path);
Прототип	direct.h
Описание	Зададената чрез path директория (тя трябва да съществува) става текущо активна.

Аргументът path може да включва и име на устройство.
Например:

chdir("a:\\turboc") или chdir("a:/turboc");

Резултат При нормално завършване chdir връща стойност нула. При грешка върнатата стойност е -1 и errno получава стойност:

ENOENT Ненамерен път или име на файл
Вж. също: chmod

Име **chmod** - променя режима на достъп до файл
Употреба **#include** <sys\stat.h>
int chmod(**char** *filename, **int** permiss);
Прототип io.h
Описание chmod определя позволения режим за достъп до файла,в съответствие с маската дадена от permiss. filename задава името на файла.
permiss може да съдържа едната или и двете символни константи: S_IWRITE и S_IREAD (дефинирани в stat.h).

Стойност на permiss	Позволен режим на достъп
S_IWRITE	писане
S_IREAD	четене
S_IREAD|S_IWRITE	четене и писане

Функцията _chmod може да прочете или да зададе атрибутите на файла за MS-DOS. Ако func е нула, функцията връща текущо действуващите атрибути на файла. При стойност на func 1, атрибутите получават стойност от atrib.

atrib може да бъде една от следните символни константи (дефинирани в dos.h):

FA_RDONLY	Достъпно само за четене
FA_HIDDEN	Скрит файл
FA_SYSTEM	Системен файл

Резултат При успешна промяна на метода за достъп до файла, chmod връща нула. В противен случай резултатът е -1.

При успешно завършване _chmod връща файловите атрибути (думата с атрибутите). В противен случай резултатът е -1.

При грешка errno получава някоя от следните стойности:

ENOENT	Ненамерен път или файл
EACCES	Достъпът отказан

Преносимост chmod е разработена за ОС UNIX.
_chmod е само за MS-DOS.

Пример:

```
#include <stdio.h>
#include <sys\stat.h>
#include <io.h>

/* Прави файла достъпен само за четене */

void make_read_only(char *filename)
{
 int stat;

 stat = chmod(filename,S_IREAD);
 if (stat)
   printf("Не е възможно да се направи %s, достъпен само за четене\n",
            filename);
 else
    printf("%s вече е достъпен само за четене\n",filename);
}

main()
{
 make_read_only("NOTEXIST.FIL");
 make_read_only("EXISTF.FIL");
}
```

Печат от работата на програмата:

Не е възможно да се направи NOTEXIST.FIL, достъпен само за четене EXISTF.FIL вече е достъпен само за четене

Име	**close** - затваря файлов манипулатор
Употреба	**int** close (**int** handle);
Други	**int** _dos_close(**int** handle);
Прототип	в io.h
Описание	close и _dos_close затварят канала (отворен за файл), указан чрез handle. Файловият манипулатор handle е получен от работата на някоя от функциите _dos_creat, creat, _dos_creatnew, creattemp, dup, dup2, _dos_open или open.
Резултат	При успешно завършване close и _close връщат нула. В противен случай връщат стойност -1. Двете функции завършват с грешка, ако параметърът handle е невалиден и задават на errno стойност:

EBADF невалиден номер на файл.

Име	**_dos_creat** - създава нов файл или отваря стар като нов. Ако файлът вече съществува, съдържанието му се изтрива но атрибутите се запазват
Употреба	include <dos.h> **int** _creat(**char** *filename, **int** attrib);
Описание	Вж. creat

Вж. също: bsearch close dup open read write

Име **creat** - създава нов файл или отваря стар като нов
Употреба include <sys\stat.h>
int creat(**char** *filename, **int** permiss);
Други **unsigned** _dos_creatnew(**char** *filename,**int** attrib,**int** handle);
Прототип в io.h
Описание creat създава нов файл или приготвя съществуващ за записване. Името на файла се сочи от указателя filename.permiss се отнася само за новосъздадени файлове.

Ако файлът съществува и е с позволен достъп за запис, creat променя дължината на файла до 0 байта, а атрибутите му остават непроменени.

creat анализира само един бит от думата, определяща режимите за достъп (в UNIX се нарича потребителски бит за разрешаване на запис). Ако този бит е 1, във файла може да се записва. При стойност 0, файлът е достъпен само за четене. Всички останали ДОС-атрибути получават стойност 0.

permiss може да заема някоя от следните стойности (дефинирани в sys\stat.h):

Стойност на permiss	Вид достъп
S_IWRITE	Разрешен запис
S_IREAD	Разрешено четене
S_IREAD|S_IWRITE	Разрешено четене и запис

Забележка: В ДОС разрешението за писане автоматично разрешава и четенето.

Файл, създаден с _dos_creat е винаги в режима, определен чрез глобалната променлива _fmode (O_TEXT или O_BINARY).

За да се създаде файл за определен режим, може да се използува open с параметър логическо ИЛИ (OR) от O_CREAT, O_TRUNC и зададен режим на записване (O_TEXT или O_BINARY). Следователно обръщение от вида:

open ("xmp",O_CREAT|O_TRUNC|O_BINARY,S_IREAD)

ще създаде двоичен файл с име XMP, достъпен само за четене. Ако xmp вече съществува, дължината му ще стане нула.

_dos_creat интерпретира attrib, като ДОС-думата за атрибутите на файла. При това обръщение всички атрибути трябва да имат стойност. Файлът винаги се отваря в двоичен режим. При успешно създаване на файл, показалецът за работа с файла се установява в началото му. Файлът е отворен, както за четене, така и за писане.

_dos_creatnew работи както _creat, с тази разлика, че ако файлът съществува, creatnew издава съобщение за грешка и го оставянепроменен.

tmpfile,creattemp работи както _creat, с тази разлика, че filename е име на път, завършващ с обратна наклонена черта (\). За файла се подбира име несъвпадащо с никое от останалите, и то се запазва в символния низ, сочен от filename. Следователно, отделеното място (сочено от filename) трябва да бъде достатъчно за запис на новополученото име. Файлът не се изтрива автоматично след завършване на програмата.

Аргументът attrib за _creat, creatnew и creattemp може да заема следните стойности (дефинирани в dos.h):

FA_RDONLY Само за четене
FA_HIDDEN Скрит файл
FA_SYSTEM Системен файл

Резултат При успешно завършване функциите връщат нов файлов манипулатор - неотрицателно цяло число. При грешка, върнатата стойност е -1.

Ако е възникнала грешка, променливата errno получава някоя от стойностите:

ENOENT Ненамерен път или име на файл
EMFILE Твърде много едновременно отворени файлове
EACCES Достъпът отказан

Име **_dos_creatnew** - създава нов файл
Употреба include <dos.h>
int creatnew(**char** *filename, **int** attrib);
Прототип io.h
Описание Вж. creat

Изчиства индикатора за грешка на файла.
Прототип: <stdio.h>
Синтаксис: **void** clearerr(FILE *stream); Функцията записва 0 в индикатора за грешка и сигнала за край на файла
Виж също: ferror, perror, feof, rewind

Име **dup** - дублира файлов манипулатор
Употреба **int** dup(**int** handle);
Други **int** dup2(**int** oldhandle, **int** newhandle);
Прототип в io.h
Описание dup и dup2 връщат нов файлов манипулатор, който има следните общи характеристики с вече съществуващия:
същия отворен файл или устройство
същия файлов указател (което означава, че промяната на файловия указател за единия файл ще промени указателя на другия.
същия метод на достъп (за четене, за запис или четене и запис).

dup връща следващия свободен файлов манипулатор. dup2 връща файлов манипулатор със стойност newhandle. Ако файлът, свързан с newhandle, е отворен в момента на обръщението към dup2, той ще бъде затворен.

handle и **newhandle** са файлови манипулатори, получени от работата на някоя от функциите creat, open, dup, dup2

Резултат При успешно завършване dup връща нов файлов манипулатор - неотрицателно цяло число. При грешка връща стойност -1.

Когато е регистрирана грешка, errno получава някоя от следните стойности:

EMFILE Твърде много едновременно отворени файлове.
EBADF Невалиден номер на файл.

Име **dup2** - дублира файлов манипулатор
Употреба **int** dup2(**int** oldhandle, **int** newhandle);
Прототип в io.h
Описание Вж. dup

Име dos_findfirst - търси файл или група файлове в директория на диск
Употреба **int** _dos_findfirst(**char** *pathname,**int** attrib, **struct** find_t *ffblк);
Прототип dos.h, dir.h
Други **int** _dos_findnext(**struct** find_t *ffblк);
Описание dos_findfirst търси в директория на диск, като използува обръщение към системната функция на MS-DOS 0x4E.

pathname е символен низ, задаващ спецификация на търсения файл (незадължително име на устройство, път и име на файл). Името на файла може да съдържа глобалните знаци ? и * (със същото значение, както в ДОС). Ако се намери подходящ файл, структурата ffblк се запълва с информация за директорията.

attrib е използуваният от MS-DOS атрибутен байт за файла. attrib може да съвпада с една от следните константи (дефинирани в dos.h):

FA_RDONLY	Само за четене
FA_HIDDEN	Скрит файл
FA_SYSTEM	Системен файл
FA_LABEL	Етикет на том
FA_DIREC	Директория
FA_ARCH	Архивен

_dos_findnext се използува за получаване на поредица от файлове, които удовлетворяват параметъра pathname, зададен чрез _dos_findfirst. ffblк е структурата, използувана и от _dos_findfirst. Тя съдържа информацията, необходима за продължаване на търсенето. При всяко обръщение към _dos_findnext се получава по едно име на файл, докато се изчерпят файловете от директорията, които отговарят на спецификацията pathname.

Форматът на структурата ffblк е следният:

```
struct  find_t  {
  char  reserved[21];  /* Резервирано за ДОС*/
  char  attrib;       /* Атрибут */
  unsigned wr_time;     /* Час на създаване */
  unsigned wr_date;     /* Дата на създаване*/
  long   size;        /* Размер на файла*/
  char  name[13];     /*Име       на файла */
};
```

Забележете, че _dos_findfirst и _dos_findnext установяват адреса DTA на MS-DOS (адресът за обмен на данни с диска), равен на адреса на find_t

Резултат _dos_findfirst и _dos_findnext връщат 0 при успешно търсене. Когато се изчерпят файловете, отговарящи на pathname или при грешка в зададеното име, функциите връщат -1 и променливата errno получава някоя от следните стойности:

ENOENT Ненамерен път или име на файл
ENMFILE Няма повече файлове

Преносимост Само за MS-DOS

Пример:

```
#include <stdio.h>
#include <dir.h>

 void main()
 {
    struct find_t ffblк;
    int done;
    printf("Списък на файловете *.*\n");
    done = _dos_findfirst("*.*",&ffblк,0);
    while(!done)
    {
       printf("  %s\n",ffblк.ff_name);
       done = _dos_findnext(&ffblк);
    }
}
```

Печат на резултатите от работата на програмата:

```
                    Списък на файловете *.*
    29.C
    BMSMALL.DAT
    CADV.C
    42.C
    ADV.C
    48.C
    EXISTF.FIL
    ANS.TXT
    DEMO
    ADS.C
```

Име	**_dos_findnext** търси файл, който отговаря на шаблона, зададен от _dos_findfirst
Употреба	**int** findnext(**struct** ffblк *ffblк);
Прототип	в dir.h
Описание	Вж. _dos_findfirst

Име	**eof** - регистрира края на файл
Употреба	**int** eof(**int** *handle);
Прототип	в io.h
Описание	eof определя дали е достигнат краят на файла,свързан с handle.
Резултат	eof връща стойност 1 при достигнат край на файл и 0 в противен случай. Стойност -1 показва, че е регистрирана грешка и errno получава значение: EBADF Невалиден номер на файл

Име **fclose** затваря файл
Употреба **int** fclose(FILE *stream);
int fcloseall(**void**);
Прототип stdio.h
Описание Проверява дали файл с даденото име съществува и дали може да бъде четен, да се записва на него или да се изпълнява. filename сочи символния низ, съдържащ името на файла.
Резултат (fclose) 0 при сполучливо или EOF при несполучливо
(fcloseall) максималния номер на затворените файлове при успешно затваряне или EOF при неуспешно
Виж също: fopen, fflush, _dos_close

Име **fclose** - затваря входно-изходен поток
Употреба **int** fclose (FILE *stream);
int fcloseall(**void**);
int fflush(FILE *stream);
int flushall(**void**);
Прототип stdio.h
Описание **fclose** затваря посочения от stream входно-изходен поток. Всички буфери, свързани с него, се изпразват преди затварянето (информацията от тях се предава по предназначение). Отделените от системата буфери се освобождават при затварянето. Буферите, отделени със setbuf и setvbuf не се освобождават автоматично.
fcloseall затваря всички отворени входно-изходни потоци сизключение на стандартните stdin и stdout.
fflush предизвиква запис на съдържанието на буфера, свързан с отворен изходен поток. Ако потокът е входен, буферът се изчиства.
flushall изчиства всички буфери, свързани с отворени входни потоци, записва съдържанието в съответните файлове и изчиства буферите, свързани с изходни потоци. Всички операции за четене след изпълнение на flushall четат нови данни в буферите от свързаните с тях файлове.
Резултат **fclose** и fflush връщат 0 при успешно завършване.
fcloseall връща броя на затворените входно-изходни потоци.
При регистрирана грешка fclose, fcloseall и fflush връщат EOF.
flushall връща цяло число - брой на отворените входни и изходни потоци.
Виж също:

Име **fcloseall** - затваря отворени входно-изходни потоци
Употреба **int** fcloseall(**void**);
Прототип в stdio.h
Описание Вж. fclose

Име **feof** - проверява състоянието "край на файл" за входно-изходен поток
Прототип: stdio.h
Употреба **int** feof(FILE *stream);
Резултат: положително число ако текущата позиция не е край на файла или 0 ако е достигнат краят.
Виж също: clearerr, ferror, perror, _eof

Име	**ferror** - проверява за наличие на грешка във входно-изходен поток
Употреба	**int** ferror(FILE *stream) **void** clearerr(FILE *stream);
Прототип	в stdio.h
Описание	**ferror** е макродефиниция за правене тест на входно- изходен поток за грешка при запис или четене. Ако индикаторът за грешка е активиран, той остава активен до обръщение към clearerr или rewind или до затваряне на потока. **clearerr** изчиства индикатора за грешка и индикатора за край на файл (дава им стойност нула). feof е макродефиниция, която прави тест на индикатора на потока stream за условието "край на файл". Ако индикаторът за край на файл е активиран, той остава в това състояние до обръщение към rewind или до затваряне на потока.
Резултат	**ferror** връща стойност различна от нула, ако е установена грешка. **clearerr** установява в начално положение индикаторите за грешка и за край на файл за потока stream. Функцията не връща резултат. feof връща резултат различен от нула, ако индикаторът за край на файл е активиран при последната операция за четене от посочения входен поток. Индикаторът за край на файл се установява в начално положение при всяка входна операция.

Име **fflush** Записва буфера в изходния поток.
Синтаксис: **int** fflush(FILE *stream);
Прототип: <stdio.h>
Резултат: 0 ако е сполучливо, ако <stream> е отворен и е разрешен за запис в противен случай връща EOF.
Виж също: fclose, _flushall, clearerr, commode.obj

Име **fgetc** Чете символ от файл.
Синтаксис: **int** fgetc(FILE *stream); **int** _fget**char**(**void**);
Прототип : <stdio.h>
Резултат:. Прочетен символ. EOF индикатор за грешка
Виж също: getc, _getw, fputc

Име fgetpos Четене и установяване на текущата позиция във файла

Синтаксис: **int** fgetpos(FILE *stream, fpos_t *pos); **int** fsetpos(FILE *stream,fpos_t *pos);
Прототип: <stdio.h>, <errno.h>
Резултат: 0 при успех или положително число при неуспех. errno: EBADF, EINVAL
Виж също: fseek, rewind, ftell

Име **fgets** Чете символен низ от файл с максимална дължина n
Синтаксис: **char** *fgets(**char** *string, **int** n, FILE *stream);
Прототип: <stdio.h>
Резултат: <string> при успех или NULL при неуспех или край на файла.
Виж също: fputs, gets, puts

Име **filelength** - определя размера на файла в байтове
Употреба **long** filelength(**int** handle);
Прототип в io.h
Описание filelength връща дължината (в байтове) на файла, свързан с файловия манипулатор handle.
Резултат При успешно завършване на операцията, filelength връща стойност тип **long** - дължината на файла в байтове. При грешка, върнатата стойност е -1 и errno получава значение: EBADF Невалиден номер на файл

Име **fileno** връща файловия манипулатор
Синтаксис: **int** fileno(FILE *stream);
Прототип: <stdio.h>
Описание fileno е макродефиниция, която връща файловия манипулатор, свързан с входно-изходния поток stream. Ако със stream са свързани повече файлови манипулатори, fileno връща този, който е присвоен на входно-изходния поток при първото му отваряне.
Резултат Връща файлов манипулатор (стойност тип **int**), свързан с потока stream.
Виж също: filelength, freopen, fopen, fstat

Прототип: <stdio.h>
Синтаксис: **int** flushall(**void**);
Резултат: Номер на отворените входно-изходни потоци.
Виж също: fflush, fclose, clearerr, commode.obj

Име **fopen** – отваря входно-изходен поток
Употреба FILE *fopen(**char** *filename, **char** *type);
FILE *fdopen(**int** handle, **char** *type);
FILE *freopen(**char** *filename, **char** *type, FILE *stream);
Прототип stdio.h
Описание **fopen** отваря файла с име filename и го свързва с входно-изходния поток stream. fopen връща указател, който се използува за идентифициране на stream в следващите операции.
fdopen свързва stream с файловия манипулатор handle (манипулаторът е получен чрез някоя от функциите creat, dup, dup2 или open). Типът на stream трябва да съвпада с режима, на който отговаря handle.

freopen затваря потока stream и отваря на негово място файла filename, като използува описанието на потока. Потокът stream се затваря независимо от това дали операцията по отварянето на файла завършва нормално. Функцията freopen е полезна при промяна на файл, свързан с някой от стандартните потоци stdin, stdout или stderr.

Символният низ type, използуван като параметър във всички оператори заема някоя от следните стойности:

r	Отваря за четене
w	Създава за писане
a	Добавяне. Файлът се отваря за режим на дописване в края му или се създава като нов, ако не съществува до момента.
r+	Отваря съществуващ файл за обновяване (четене и запис)
w+	Създава нов файл за запис и четене
a+	Отваря за добавяне. Отваря (или създава, ако файлът не съществува) за дописване в края на файла.

За да се зададе, че определен файл е отворен или създаден за текстов режим, обикновено се добавя t към type(напр. rt, w+t, ...). Аналогично, за да се зададе двоичен режим се използува наставката b (напр. wb, a+b, ...).

Ако видът не е указан с t или b, той се определя от глобалната променлива _fmode. Ако _fmode има стойност O_BINARY, файловете ще се отварят като двоични. Ако _fmode има стойност O_TEXT, те ще бъдат отваряни като текстови. Константите O_... са дефинирани в заглавния файл fcntl.h.

Когато файл е отворен за обновяване, за резултантния поток са позволени както входни, така и изходни операции, но трябва да се спазват следните условия: изходна операция не може веднага да се следва от входна. Те трябва да са разделени от обръщение към функция fseek или rewind. Респективно изходна операция трябва да бъде отделена от следваща я входна чрез обръщение към някоя от функциите fseek или rewind или входна операция, която регистрира условието за достигнат край на файл.

Резултат При успешно завършване всяка функция връща нов отворен входно-изходен поток. freopen връща аргумента stream. При грешка всяка функция връща NULL.

Пример:

```
#include <stdio.h>
#include <fcntl.h>  /* Необходимо за дефиниране на режима за  open*/

  void main()
  {
    int handle, status;
    FILE *stream;

    /* Отваряне на файла */
    handle = open("MYFILE.TXT",O_CREAT);

    /* Включване във входно-изходен поток */
    stream = fdopen(handle,"w");
    if (stream==NULL)
       printf("Грешка при fdopen\n");
    else {
```

```
    fprintf(stream,"Привет!\n");
    fclose(stream);
    }
}
```

Име **fprintf** - изпраща към входно-изходен поток (stream) аргументите си (arg...), записани съгласно спецификациите във форматиращия низ (**for**mat).
Употреба **int** fprintf(FILE *stream, **char** ***for**mat [,arg, ...]);
Прототип в stdio.h
Описание Вж. printf
Виж също: printf, fscanf, _cprintf, sprintf

Име **fputc** Записва байт във файл
Прототип: <stdio.h>
Синтаксис: **int** fputc(**int** c, FILE *stream);
Резултат: Записан символ. EOF индицира за грешка.
Виж също: fgetc, putc

Име **fputs** Записва низ във файл
Прототип: <stdio.h>
Синтаксис: **int** fputs(**char** *string, FILE *stream);
Резултат: Положително число при успех или EOF при грешка.
Виж също: fgets, puts, gets

Име **fread** Чете зададен брой байтове от файл.
Прототип: <stdio.h>
Синтаксис: size_t fread(**void** *buffer, size_t size, size_t count, FILE *stream);
Описание: *buffer - Буфер съдържащ прочетената информация
size- Размер на елементите count - Брой елементи за четене
Резултат: Брой прочетени байтове.
Виж също: fwrite, _read

Име **freopen** - затваря входно-изходен поток и отваря нов файл като използува описанието на потока
Прототип: <stdio.h>
Синтаксис: FILE *freopen(**char** *filename,**char** *mode,FILE *stream); mode: "r", "w", "a", "r+", "w+", "a+" ("t" or "b" appended to <mode> to indicate type)
Резултат: Указател към новия отворен файл при успех или NULL при неуспех.Функцията е удобна за пренасочване на стандартните файлове stdin, stdout, stderr, stdaux, stdprn
Виж също: _dup, fopen, _fileno

Име **fscanf** - чете данни от входно-изходен поток в съответствие със зададен форматиращ низ

Прототип: <stdio.h>
Синтаксис: **int** fscanf(FILE *stream,**char *for**mat [,argument]...);
Резултат: Брой на успешно конвертираните данни . EOF индицира за грешка или за достигане на края на файла.
Виж също scanf, fprintf, _cscanf, sscanf

Име	**fseeк** - изменя текущата позиция (за писане или четене) във входно-изходен поток
Употреба	**int** fseeк(FILE *stream, **long** offset, **int** fromwhere); fromwhere: SEEK_CUR, SEEK_END, SEEK_SET **long** ftell(FILE *stream); **void** rewind(FILE *stream);
Прототип	в stdio.h
Виж също:	ftell, _lseeк, rewind, fgetpos, fsetpos
Описание	**fseeк** променя стойността на указателя, свързан с входно-изходния поток stream, което е равносилно на нова сочена от него позиция. Новата позиция в потока е на offset на брой байта от мястото, зададено от fromwhere. **fromwhere** трябва да заема стойности 0, 1 или 2, които съответстват на трите символни константи (дефинирани в stdio.h): fromwhere Място във файла SEEK_SET (0) Начало SEEK_CUR (1) Текущата позиция SEEK_END (2) край **fseeк** пренебрегва символите, върнати обратно чрез ungetc. **Ftell** връща текущата позиция на указателя в stream. В случая offset е отместването в байтове от началото на файла. rewind(stream) работи еквивалентно на fseeк(stream, 0L,SEEK_SET) с тази разлика, че rewind изчиства индикаторите за край на файл и за грешка, докато fseeк изчиства само индикатора за край на файл. След fseeк или rewind може да следва входна или изходна операция.
Резултат	При успешно придвижване на указателя, fseeк и rewind връщат стойност 0, а при грешка - стойност различна от нула. При нормално изпълнение ftell връща текущата позиция на указателя във файла, а при грешка - стойността е -1L.

Пример:

```
   #include <stdio.h>
/* Връща броя байтове във входно-изходния поток */
   long filesize(FILE *stream)
   {
         long curpos, length;
          curpos = ftell(stream);
         fseeк(stream,0L,SEEK_END);
         length = ftell(stream);
         fseeк(stream,curpos,SEEK_SET);
         return(length);
     }

     void main()
     {
         FILE *stream;
         stream = fopen("MYFILE.TXT","r");
         printf("Размерът на MYFILE.TXT е %ld байта\n",
```

```
        filesize(stream));
    }
```

Печат от работата на програмата:

Размерът на MYFILE.TXT е 9 байта

Име **fstat** - дава информация за отворен файл
Употреба include <sys\stat.h>**int** fstat(**int** handle, **struct** stat *buff);
Прототип в sys\stat.h
Описание Вж. stat

Име **fsopen** Отваря входно-изходен поток
Прототип: <stdio.h>, <share.h>
Синтаксис: FILE *_fsopen(**char** *filename,**char** *mode,**int** shflag);
Режими : "r", "w", "a", "r+", "w+", "a+" ("t" или "b" метод за отваряне на файла)
Shflag: _SH_COMPAT, _SH_DENYNO, _SH_DENYRD, _SH_DENYRW, _SH_DENYWR
Резултат: Указател към потока при успех или NULL при неуспех
Виж също: fopen, fclose, _sopen, _locking, ferror

Име **ftell** Текуща стойност на указателя на файла.
Прототип: <stdio.h>
Синтаксис: **long** ftell(FILE *stream);
Резултат: Текуща позиция във файла или -1L при грешка errno: EBADF, EINVAL
Виж също: fgetpos, fseek, _lseek, _tell

Име **fwrite** - записва определен брой байтове в изходния поток
Синтаксис: **int** fwrite(**char** *str,size_t dd,size_t count, FILE *stream);
size_t е дефинирано като цяло без знак (**unsigned**)
Прототип: <stdio.h>
Резултат: Брой на правилно записаните байтове.
Виж също: fread, _write

Име **getc** Чете байт от файл.
Прототип: <stdio.h>
Синтаксис: **int** getc(FILE *stream);
Резултат: Прочетен символ. EOF при грешка или край на файла.
Виж също: fgetc, _getw, putc, ungetc, _getch

Име **gets** Чете стринг от файл.
Прототип: <stdio.h>
Синтаксис: **char** *gets(**char** *buffer);
Резултат: Указател към стринга или NULL при грешка или край на файла.
Виж също: fgets, fputs, puts, _cgets

Име **getw** Чете дума от файл
Прототип: <stdio.h>
Синтаксис: **int** _getw(FILE *stream);
Резултат: Цяло число .EOF индикатор за грешка или край на файл.
Виж също: _ putw, getc, fgetc

Име **putc** Записва символ във файл.
Прототип: <stdio.h>
Синтаксис: **int** putc(**int** c, FILE *stream);
c - символ за запис
*stream - указател към структура описваща състоянието на файла за запис
Резултат: Записан символ. EOF индикатор за грешка.
Виж също: fputc, _putw, getc

Име **getcwd** - установява текущо активната работна директория
Употреба **char** *getcwd(**char** *buf, **int** buflen);
Прототип в dir.h
Описание getcwd дава пълната спецификация (устройство, път и име) на текущата работна директория.
Допустимата дължина на получения низ е buflen (дължината на буфера). Името се съхранява в мястото, сочено от указателя buf.
Ако дължината на спецификацията е по- дълга от buflen, възниква грешка.
Ако buf е NULL, системата сама ще използува функция malloc за да отдели буфер с дължина buflen. По-късно програмистът сам може да освободи паметта за буфера като използува функция free суказателя, който връща getcwd.
Резултат getcwd връща указателя buf.
При грешка върнатата стойност е NULL и глобалната променлива errno заема някоя от стойностите:
ENODEV Няма такова устройство
ENOMEM Няма достатъчно памет
ERANGE Резултат извън обхвата

Име **_dos_getdiskfree** - дава свободното място на диск
Употреба **void** _dos_getdiskfree(**unsigned char** drive, **struct** diskfree_t *dfreep);
Прототип в dos.h

Описание **_dos_getdiskfree** приема спецификация на устройство чрез drive (0 = подразбиращото се, 1 = A, ...) и запълва структурата, сочена от dfreep, с характеристиките на диска.

Структурата diskfree_t е дефинирана по следния начин:

```
struct  diskfree_t  {
  unsigned avail_clusters;       /* Свободни кластери */
  unsigned total_clusters;       /* Общ брой кластери */
  unsigned sectors_pen_clusters; /* Сектори в кластер */
  unsigned bytes_pen_sectors;    /* Байтове в сектор  */
 };
```

ЗАБЕЛЕЖКА: Кластерите са група от сектори (броят на секторите зависи от обема на диск(етат)а. В нашата литература със същото значение се използува и думата "пакет".

Резултат **_dos_getdiskfree** връща 0 при липса на грешка. В случай на грешка връща номера на в структурата dfree получава стойност -1.

Вж. също: getcurdir setdisk

Име	**_dos_getdrive** - установява текущо активното дисково устройство
Употреба	**void** _dos_getdrive(**unsigned** *drive); **void** _dos_setdrive(**unsigned** drive,**unsigned** *brdrive);
Прототип	в dir.h
Описание	**_dos_getdrive** установява текущо активното дисково устройство и връща 0 - за устройство A, 1 за B, 2 за C и т.н. (Еквивалентно на ДОС-функцията 0x19). **_dos_setdrive** задава кое да бъде текущо активното дисково устройство чрез drive (0 = A, 1 = B, 2 = C, ...). Това е еквивалентно на работата на ДОС-функцията 0x0E.
Резултат	**_dos_getdrive** връща текущо активното дисково устройство. **_dos_setdrive** връща броя на достъпните дискови устройства.

Име _ **dos_getftime** - установява датата и часа на създаването на файл

Употреба **int** _dos_getftime(**int** handle,**unsigned** *date, **unsigned** *time);
int _dos_setftime(**int** handle,**unsigned** date, **unsigned** time)

Прототип в io.h

Описание **_dos_getftime** установява часа и датата за дисков файл, свързан с файловия манипулатор handle. Данните се запълват в date а часът в time.
_dos_setftime заменя часа и датата на дисков файл свързан с файловия манипулатор handle с данните от date и time

Резултат Двете функции при успешно завършване връщат нула.
В случай на грешка връщат -1 и глобалната променлива errno заема някоя от следните стойности:
EINVFNC - Невалиден номер на функция
EBADF - Невалиден номер на файл

Име **mkdir** - създава директория

Употреба **int** mkdir(**char** *pathname);
int rmdir(**char** *pathname);

Прототип в dir.h

Описание **mkdir** взема зададената спецификация pathname и създава нова директория с това име.

rmdir изтрива директорията, определена от спецификацията pathname. При това трябва да са изпълнени следните условия: директорията да е празна директорията да не е текущо активна това да не е главната директория.

Резултат **mkdir** връща стойност нула, ако новата директория е създадена.

rmdir връща стойност нула, ако директорията е изтрита.

При грешка функциите връщат -1 и errno получава една от следните стойности:

EACCES Действието е отказано

ENOENT Ненамерен път или име на файл

Име **mktemp** - създава неизползувано до момента име за файл

Употреба **char** *mktemp(**char** *template);

Прототип в dir.h

Описание mktemp замества template с ново, неизползувано до момента име на файл и връща адреса на template.

template трябва да бъде нулево прекъснат символен низ с шест поредни символа X в края, които след това се заместват с неизползувана до момента комбинация от латински букви плюс точка така, че се получава ново име на файл, с основна част от две букви, следвани от точка и разширение от три букви. Процедурата започва с предложение за име AA.AAA и целта е да се получи име, което не съвпада с име на файл от диска.

Резултат Ако template е с правилен формат, mktemp връща адреса на символния низ template. В противен случай файл не се създава или отваря.

Име **open** - отваря файл за четене или писане

Употреба **int** open(**char** *pathname, **int** access[,**int** permiss]);

int _open(**char** *pathname, **int** access);

Прототип в io.h, <fcntl.h>

Описание open отваря файл, зададен чрез pathname, след което го приготвя за четене и/или за запис, според режима, посочен чрез access.

За open, параметърът access се съставя чрез побитово OR на флаговете от следващите списъци. От първия списък може да бъде използуван само един флаг, а флаговете от останалите списъци могат да бъдат използувани във всякакви логически комбинации.

Списък 1: Флагове за режима за четене/запис:

O_RDONLY Отваря само за четене

O_WRONLY Отваря само за запис

O_RDWR Отваря за четене и запис.

Списък 2:

O_NDELAY Не се използува. Оставен за съвместимост с UNIX.

O_APPEND Указателят на файла се поставя в края му преди всяка операция за запис

O_CREAT Ако файлът съществува, флагът няма ефект. Ако файлът не съществува, той се създава и битовете, зададени чрез permiss, се използуват за определяне на атрибутите, аналогично на chmod.

O_TRUNC Ако файлът съществува, дължината му се променя на 0 (нула). Атрибутите на файла остават непроменени.

O_EXCL Не се използува. Оставен за съвместимост с UNIX.

O_BINARY Използува се за явно отваряне на файл в двоичен режим.

O_TEXT Използува се за явно отваряне на файл в текстов режим. Ако не е зададен нито O_BINARY нито O_TEXT, файлът се отваря в режима на предаване, зададен чрез глобалната променлива _fmode.

Ако при съставянето на access е използуван флагът O_CREAT, то може да се състави и използува незадължителният аргумент permiss чрез следните константи: Стойност на permiss Позволен достъп

S_IWRITE Позволен запис

S_IREAD Позволено четене

S_IREAD|S_IWRITE Позволени четене и запис

За open, стойността на access за MS-DOS 2.x се ограничава от O_RDONLY, O_WRONLY и O_RDWR.

За MS-DOS 3.x се допускат и следните стойности:

O_NOINHERIT Файлът не може да се предава на породен процес

O_DENYALL Позволява само текущо активният манипулатор да има достъп до файла

O_DENYWRITE Позволява само четене от всички други отворени файлове.

O_DENYREAD Позволява само запис за всички други отворени файлове.

O_DENYNONE Позволява съвместно използуване на файла (отваряне (share) от други процеси).

Оператор _open в комбинация с MS-DOS 3.x може да използува само една от стойностите O_DENYxxx. Тези режими се отнасят за съвместното използуване на файлове (file-sharing) и са добавка към другите начини за определяне достъпа до файл. Максималният номер на едновременно отворени файлове зависи от конфигурационните параметри на системата.

Резултат При успешно завършване, функциите връщат неотрицателно цяло число (файлов манипулатор) и указателят на файла, показващ текущата позиция, се премества в началото на файла. При грешка, двете функции връщат -1 и задават на errno някоя от следните стойности:

ENOENT Ненамерен път или име на файл

EMFILE Твърде много отворени файлове EACCES Операцията отказана

EINVACC Неправилен код за достъп режими се отнасят за съвместното използуване на файлове (file-sharing) и са добавка към другите начини за определяне достъпа до файл.

Максималният номер на едновременно отворени файлове зависи от конфигурационните параметри на системата.

Име **_putw** Записва дума във файл.

Прототип: <stdio.h>

Синтаксис: **int** _putw(**int** bin**int**, FILE *stream);

bininit - символ за запис

*stream W- указатл към структура описващасъстиоянието на файла за запис

Резултат: Записана стойност. EOF при грешка.

Виж също: _getw, putc, fputc

Име **read** - чете от файл
Употреба nt read(**int** handle, **void** *buf, **unsigned** nbyte);
Прототип io.h
Описание **read** прави опит да четат nbyte на брой байтове от файла, свързан с файловия манипулатор handle в буфера, сочен от buf.
За файл отворен в текстов режим, read маха символа за преминаване в началото на реда (CR) и регистрира край на файл при прочитане на символ Ctrl-Z. handle е файлов манипулатор, получен чрез creat, open,dup, dup2 или fcntl.
При дискови файлове функцията започват да чете от текущото положение на указателя. След завършване на четенето, файловият указател се премества с прочетения брой байтове. При други устройства, байтовете се четат направо от тях.
Резултат При успешно завършване функцията връща цяло положително число, показващо броя на записаните в буфера байтове. Ако файлът е бил отворен в текстов режим, read не брои символите за край на ред (CR) или края на файла (Ctrl-Z). При достигане на край на файл функцията връща нула.
При грешка резултатът е -1 и errno получава някоя от следните стойности:
EACCES Достъпът отказан
EBADF Невалиден номер на файл

Име **rewind** - премества указателя за входно-изходен поток в началото
Прототип: <stdio.h>
Синтаксис: **void** rewind(FILE *stream);
Виж също: fseek, ftell, fsetpos, feof, seek

Име **_rmtmp** Затваря и изтрива временните файлове в текущата директория
Прототип: <stdio.h>
Синтаксис: **int** _rmtmp(**void**);
Резултат: Брой на затворените и изтрити временни файлове.
Виж също: tmpfile, tmpnam

Име **rmdir** - премахва директория
Употреба **int** rmdir(**char** *pathname);
Прототип в dir.h
Описание Вж. mkdir

Име **setbuf** - присвоява буфер на входно-изходен поток
Употреба **void** setbuf(FILE *stream, **char** *buf);
int setvbuf(FILE *stream,**char** *buf,**int** type, size_t size);
size_t е тип дефиниран като **unsigned**
type: _IOFBF, _IOLBF, _IONBF
Прототип в stdio.h
Описание **setbuf** и **setvbuf** назначават буфера buf да бъде използуван за буфериране на вход и изход, вместо автоматично отделяния за целта буфер. Използуват се след отварянето на входно-изходния поток. При setbuf, ако buf е NULL,

съответните входни и изходниоперации ще работят без буфериране, а ако buf не е NULL – ще работят с пълно буфериране.

Буферът трябва да е с дължина BUFSIZ байта (BUFSIZ е дефинирана в stdio.h). При setvbuf, ако buf е NULL, буферът ще бъде отделен като се използува функция malloc. Дължината ще се определя от аргумента size, който трябва да бъде по-голям от нула. stdin и stdout са небуферирани, когато не са пренасочени. Ако се пренасочат, те работят като пълно буферирани. Режимът на буфериране в този случай може да се промени с помощта на setbuf.

"Небуфериран" означава, че символите записвани в потока се предават веднага на файла или устройството, докато "буферирането" означава, че символите се събират и се записват на порции. Във функция setvbuf, параметърът type заема някоя от следните стойности:

_IOFBF Файлът е пълно буфериран. При празен буфер, следващата входна операция ще се опита да запълни целия буфер. При изходна операция, буферът трябва да бъде запълнен изцяло преди да се запишат данни във файла.

_IOLBF Файлът е буфериран по редове. При празен буфер, следващата входна операция ще се опита да го запълни изцяло. При изходна операция обаче, буферът ще бъде изпразнен, когато във файла се запише символ за нов ред.

_IONBF Файлът не е буфериран. Параметрите buf и size се игнорират. Всяка входна операция чете директно от файла и всяка изходна операция пише незабавно в него. setbuf може да бъде извикана за входно-изходен поток само веднага след отварянето му или след обръщение към fseek. В противен случай резултатите са непредвидими. Обръщение към setbuf след като входно- изходен поток е бил буфериран е позволено и не предизвиква проблеми. Честа причина за грешки е отделянето на буфер като автоматична (локална) променлива и пропускане затварянето на файла преди връщане от съответната функция.

Резултат setbuf не връща резултат. setvbuf връща нула при нормално завършване и стойност различна от нула, ако е регистрирана невалидна стойност на type или size, ако buf е NULL или няма достатъчно място за буфера

Пример:

```
#include <stdio.h>

void main()
{
  FILE *input, *output;
  char bufr[512];

  input = fopen("file.in","r");
  output = fopen("file.out","W");

  /* Използуване на собствен буфер за входния поток */

  if (setvbuf(input,bufr,_IOFBF, 512) != 0)
    printf("Не е отделен буфер за входен файл\n");
  else
    printf("Отделен е буфер за входен файл\n");

 /* Настройка на изходен поток за буфериране по редове, като се
```

```
    използува мястото, получено чрез непряко обръщение към
    malloc*/
  if (setvbuf(output,NULL,_IOLBF,132) != 0)
    printf("Неуспешен опит за буфериране на изходен файл\n");
  else
    printf("Отделен е буфер за изходен файл\n");

  if (setvbuf(input,bufr,_IOFBF, 512) != 0)
    printf("Не е отделен буфер за входен файл\n");
  else
    printf("Отделен е буфер за входен файл\n");

/* Настройка на изходен поток за буфериране по редове, като се
използува мястото, получено чрез непряко обръщение към     malloc*/
  if (setvbuf(output,NULL,_IOLBF,132) != 0)
    printf("Неуспешен опит за буфериране на изходен файл\n");
  else
    printf("Отделен е буфер за изходен файл\n");
  /* Вход/Изход от файл.  Затваряне на файла */
  fclose(input);
  fclose(output);
  }
}
```
също: getcbrк

Име **_dos_setdrive** - прави зададеното устройство текущо активно
Употреба **int** _dos_setdrive(**int** drive,**int** *brdrive);
Прототип в dir.h
Описание Вж. _dos_getdruive

Име **_dos_setftime** - поставя часа и датата на файл
Употреба **int** _dos_setftime(**int** handle,**unsigned** date, **unsigned** time)
Прототип в io.h
Описание Вж. _dos_getftime

Име **setmode** - задава режима на отворен файл
Употреба **int** setmode(**int** handle, **int** mode);
Прототип в io.h
Описание **setmode** задава режима (двоичен или текстов), за отворен файл, свързан с файловия манипулатор handle). Аргументът mode трябва да има стойност O_BINARY или O_TEXT.
Резултат setmode връща нула при успешно завършване.
При грешка, върнатата стойност е -1 и errno получава стойност:
EINVAL Невалиден аргумент

Име **setvbuf** - присвоява буфер на входно-изходен поток
Употреба **int** setvbuf(FILE *stream,**char** *buf,**int** type, size_t size);
size_t е тип дефиниран като **unsigned**, но въпреки това максималната стойност за size е 32767!
Прототип в stdio.h

Име **tmpfile** Създава временен файл.
Прототип: <stdio.h>
Синтаксис: FILE *tmpfile(**void**);
Резултат: Указател към файла при успех или NULL при грешка.
Виж също: _rmtmp, _tempnam

Име **stat** - дава информация, свързана с отворен файл
Употреба **int** stat(**char** *pathname, **struct** stat *buff); **int** fstat(**int** handle, **struct** stat *buff);
Прототип в sys\stat.h
Описание stat и fstat съхраняват информация за определен отворен файл (или директория) в структурата stat.
stat събира информация за отворен файл (или директория) определен от pathname.
fstat събира информация за отворен файл, свързан с файловия манипулатор handle.

И при двете функции buff сочи структурата stat (дефинирана в sys\stat.h), която съдържа следните полета:

st_mode маска от битове, които дават информация за режима, в който е отворен файлът.
st_dev номер на дисково устройство, съдържащо файла или файлов манипулатор, ако файлът е на устройство.
st_rdev също както st_dev
st_nlinк равно на цялата константа 1
st_size размер на отворения файл в байтове
st_atime час на последната модификация на отворения файл
st_mtime също както st_atime
st_ctime също както st_atime

Структурата stat съдържа още три полета, но в тях се пазят стойности, които нямат смисъл за MS-DOS.

Маската с информация за режима на отворения файл включва поредица от битове, като един от следните битове има стойност 1:

S_IFCHR зададен (стойност 1), ако handle се отнася за знаково устройство (fstat)
S_IFREG зададен (стойност 1), ако handle се отнася за същински файл (fstat) или файлът е зададен чрез pathname (stat)
S_IFDIR зададен (стойност 1), ако pathname специфицира директория (stat)

Един или два от следните битове имат стойност 1:
S_IWRITE зададен (стойност 1), ако потребителят

Един или два от следните битове имат стойност 1:
S_IWRITE зададен (стойност 1), ако потребителят има право да прави записи във файла

S_IREAD зададен (стойност 1), ако потребителят има право да чете от файла

За stat, маската включва и битове, които се задаватв зависимост от разширението на отворения файл(битове, определящи режима на изпълнение на файла за потребителя).

Резултат Двете функции връщат нула при успешно намерена и съхранена информация за отворения файл. При грешка, резултатът е -1 и errno получава стойност:

ENOENT Ненамерен път или файл (за stat)
EBADF Невалиден манипулатор за файл (за fstat)

Име tell - установява текущата позиция на файлов указател
Употреба **long** tell(**int** handle);
Прототип в io.h
Описание Вж. fseek

Име **_tempnam** Създава име на временен файл
Прототип: <stdio.h>
Синтаксис: **char** *_tempnam(**char** *dir, **char** *prefix);
char *tmpnam(**char** *string);
Резултат: указател към новото име на файла или NULL при грешка.
Виж също: tmpfile

Име **vfprintf** Форматен запис във файл. Препоръчват се при програмиране под Windows.
Прототип: <stdio.h>, <stdarg.h>, <varargs.h>
Синтаксис: **int** vfprintf(FILE *stream,**char** ***for**mat,va_list argptr);
int vprintf(**char** ***for**mat, va_list argptr);
int vsprintf(**char** *buffer,**char** ***for**mat,va_list argptr);
int _vsnprintf(**char** *buffer,size_t count,**char** ***for**mat, va_list argptr);
Резултат: брой на записаните байтове при успех или отрицателночисло при грешка.
Виж също: Printf, fprintf, sprintf, _cprintf, va_arg

```
/* Тази примерна програма демонстрира действието на функциите
    fopen  fread  fwrite  feof fclose
  Програмата презаписва един файл в друг като извежда на екрана
  съдържанието му.*/
#include <stdio.h>
FILE *input,*otput;
 char str[81];
void main(int argc,char *argv[])
{
 if(argc < 3 )
  { printf(" \n Малко параметри\n");
    exit(0);
  }
 if((input=fopen(argv[1],"rb")) == NULL)
    { printf("\n Не е намерен файл %s \n",argv[1]);
      exit(0);
```

```
  }
 else {
  if((output=fopen(argv[2],"wb")) == NULL)
    { printf("\n Не е отворен файл %s \n",argv[2]);
      fclose(input);
      exit(0);
  }
     else {
      while(!feof(input))
      {
        fread(str,1,80,input);
        printf("%s",str);
        fwrite(str,1,80,output);
      }
    fclose(input);
    fclose(output);
    }
  }
}
```

Име **_makepath** - съставя спецификация на файл (път и име)

Употреба **void** _makepath(**char** *path, **char** *drive, **char** *dir,
char *name, **char** *ext);

Други **int** _splitpath(**char** *path, **char** *drive, **char** *dir,**char** *name, **char** *ext);

Прототип в dir.h

Описание _makepath съставя спецификация на файл от отделните компоненти. Получената спецификация има вида:

X:\DIR\SUBDIR\NAME.EXT

където

X се определя от drive
\DIR\SUBDIR\ се определя от dir
NAME се определя от name
EXT се определя от ext

_splitpath обработва спецификация на файл, като я разбива наотделни компоненти. За разгледания пример fnsplit ще получи отзададената спецификация компонентите drive, dir, name и ext.

Максималните размери за тези символни низове се дават от константите MAXDRIVE, MAXDIR, MAXPATH, MAXNAME и MAXEXT (дефинирани в dir.h). Всеки размер включва място за завършващия символ (\0) за низа.

Константа	Макс.размер	Символен низ
MAXPATH	80	path
MAXDRIVE	3	drive (вкл. ":")
MAXDIR	66	dir (вкл. символ "\" в началото и края)
MAXFILE	9	name
MAXEXT	5	ext (вкл. точката в началото)

_splitpath приема, че има достатъчно място за запазването на всяка компонента. fnmerge приема, че има място за запазване на конструираната спецификация (максимално възможната й дължина се определя от MAXPATH).

При работата си _splitpath третира пунктуацията по следния начин:
drive взема символа ":" (напр. A:,C:...)
dir взема водещия и завършващия символ "\" (напр.\user\, \text\ )
ext взема точката в началото на разширението (напр. .c, .txt)

Пример:

```
#include <stdio.h>
#include <dir.h>

char drive[MAXDRIVE];
char dir[MAXDIR];
char file[MAXFILE];
char ext[MAXEXT];

main()
{
  char s[MAXPATH], t[MAXPATH];
  int flag;

  for (;;) {
    printf("> "); /* Печата знака за готовност на ДОС */
    if (!gets(s)) break;  /* Докато няма повече въвеждане */
    _splitpath(s,drive,dir,file,ext);
    printf(" Диск: %s, Директория: %s, Файл: %s, Разширение: %s, ", drive, dir, file, ext);
    printf("Флагове: ");
    if (flag & DRIVE)           printf(":");
    if (flag & DIRECTORY)  printf("d");
    if (flag & FILENAME)     printf("f");
    printf("d");
    if (flag & FILENAME)      printf("f");
    if (flag & EXTENSION)     printf("e");
    printf("\n");
    fnmerge(t,drive,dir,file,ext);
    if (strcmp(t,s) !=0)  printf("--> Символните низове са различни");
  }
}
```

Име **_splitpath** - разделя пълната спецификация на файл на отделни компоненти (устройство, директории, име, разширение)
Употреба **void** _splitpath(**char** *path, **char** *drive, **char** *dir, **char** *name, **char** *ext);

Прототип dir.h
Описание Вж. _makepath

Име Unlink - изтрива файл
Употреба **int** unlink(**char** *filename);
Прототип в dos.h
Описание Unlink изтрива файла, зададен чрез filename. filename може да съдържа

пълна спецификация на файл (устройство, път и име). Глобалните символи (* и ?) не са позволени. Файлове достъпни само за четене не могат да бъдат изтрити с помощта на функцията. За тях първо трябва да се използува функция chmod или _chmod за да се направи файлът достъпен.

Резултат При успешно завършване, функцията връща нула.
При грешка, резултатът е -1 и errno получава някоя от стойностите:
ENOENT Ненамерен път или файл
EACCES Достъпът отказан

Име Write - записва във файл
Употреба **int** write(**int** handle, **void** *buf, **int** nbyte);
Прототип io.h
Описание Write е функция, които записват данните от буфера, сочен от buf във файл или устройство, определено от handle. handle е файлов манипулатор, получен чрез някоя от функциите creat, open, dup, dup2 .
Функцията прави опит да запишат във файла, свързан с handle, nbyte на брой байта от буфера, сочен от buf. С изключение на случая при запис в текстов файл, количеството записани байтове не надхвърля заявеното чрез nbyte. При текстови файлове, когато write срещне символа за нов ред (LF), тя прави запис на комбинацията от символи за преминаване в началото на реда и след това на следващият ред (CR/LF). Записването на по-малък брой байтове от заявения е признак за възникнала грешка, вероятно носителят е пълен. При работа с файлове на диск, записването винаги продължава от текущото положение на указателя за файла (вж. lseek). В останалите случаи, байтовете се отпращат направо към съответното устройство. За файлове, отворени с опция O_APPEND, write позиционира указателя на файла в края му преди запис.

Резултат Функцята връща броя на записаните байтове. При запис в текстов файл не се смята символа за продължава от текущото положение на указателя за файла (вж. lseek). В останалите случаи, байтовете се отпращат направо към съответното устройство. За файлове, отворени с опция O_APPEND, write позиционира указателя на файла в края му преди запис.При запис в текстов файл не се смята символа за преминаване в началото на реда (CR).При грешка функциите връщат стойност -1 и errno получава някоя от следните стойности:
EACCES Достъпът отказан
EBADF Невалиден номер на файл

```
//Прави разбор на  името на  файл предаван пакато параметър от командния ред.

#include <stdlib.h>
#include <stdio.h>
int main( int argc, char *argv[], char *envp[])
{
  char path_buffer[_MAX_PATH];
  char drive[_MAX_DRIVE];
  char dir[_MAX_DIR];
  char fname[_MAX_FNAME];
  char ext[_MAX_EXT];
```

```
   if ( argc < 1){ printf("\nSyntax: %s FileName\n\n",argv[0]); return 1; }
   _splitpath( argv[1], drive, dir, fname, ext );
   printf( "Path extracted with _splitpath:\n" );
   printf("File name: %s\n",argv[1]);
   printf( "  Drive: %s\n", drive );
   printf( "  Dir: %s\n", dir );
   printf( "  Filename: %s\n", fname );
   printf( "  Ext: %s\n", ext );
}

// Пример за работа с файл  с пряк достъп
// Преобразува символите на кирилица в даден от
#include  <stdio.h>
int main( int argc, char *argv[], char *envp[])
{
  char Buff[1000];
  unsigned int BrSize,i;
  long Size=0;
  char c1,c2;
  FILE *fp;
  if( argc < 2) {printf("\nSyntax : %s  FileName",argv[0]); return 1; }
  if(( fp=fopen(argv[1],"r+b")) == NULL){ printf("\n Error open file %s",argv[1]);return 2;}
  printf("\n\a\a FileName %s",argv[1]);
  c1 = (char)0xA0;
  c2 = (char)0xD0;
  do {
    BrSize=fread(Buff,1,1000,fp);
    printf("\nBrSize = %d",BrSize);
    for(i=0;i<BrSize;i++)
        if ( Buff[i] >= c1 && Buff[i] <c2)   Buff[i] = Buff[i] - ' ';
    fseek(fp,Size,SEEK_SET);
    fwrite(Buff,1,BrSize,fp);
    Size += BrSize;
  }
  while(BrSize == 1000);
  fclose(fp);
}




// Извежда на екрана съдържанието на заглавния блок на EXE програма.

#include <stdio.h>
struct HeadExe
{
  unsigned int idExe; //5A4D
  unsigned int LenMod; // Дължина
  unsigned int LenDiv;//   Дължина
  unsigned int BrSegAdr;//
  unsigned int SizeHead;//
  unsigned int MemMin;//
  unsigned int MemMax;//
  unsigned int OffStack;//
  unsigned int RegSp;//
  unsigned int CheskWord;//
  unsigned int RegIP;//
  unsigned int  CodeSeg;//
  unsigned int AdrTable;//
  unsigned int BrOverlai;//
```

```
        };

int main( int argc, char *argv[], char *envp[])
{
    FILE *in;
    struct HeadExe Buff;
     if ( argc < 2 )  {  printf("\nSyntax : %s   File_Name_In ",argv[0]);  return 1; }
     if((in=fopen(argv[1],"rb"))==NULL)  { printf("\nError open file %s",argv[1]); return 2;  }
    if( fread(&Buff,1,sizeof(struct HeadExe),in) )
    {
        printf("\nАдрес| Съдържание                                    | Тип  | Стойност ");
        printf("\n00H|Идентификатор на прог. от тип EXE(5A4DH)              |1 дума|%4x",Buff.idExe);
        printf("\n02H|Дължина на файла MOD 512                           |1 дума|%u",Buff.LenMod);
        printf("\n04H|Дължина на файла DIV 512                          |1 дума|%u",Buff.LenDiv);
        printf("\n06H|Броя на сегментните адреси за пренасочване            |1 дума|%u",Buff.BrSegAdr);
        printf("\n08H|Размер на заглавния блок в параграфи                |1 дума|%u",Buff.SizeHead);
        printf("\n0AH|Минимален брой необходими параграфи                  |1 дума|%u",Buff.MemMin);
        printf("\n0CH|Максимален брой необходими параграфи                 |1 дума|%u",Buff.MemMax);
        printf("\n0EH|Изместването на стековия сегмент                    |1 дума|%u",Buff.OffStack);
        printf("\n10H|Съдържание на регистъра SP при стартиране на програмата  |1 дума|%u",Buff.RegSp);
        printf("\n12H|Контролна сума,базирана на заглавната част на EXE файла  |1 дума|%u",Buff.CheskWord);
        printf("\n14H|Съдържанието на регистъра IP при стартиране на програмата|1 дума|%u",Buff.RegIP);
        printf("\n16H|Началото на кодовия сегмент в EXE файла              |1 дума|%u",Buff.CodeSeg);
        printf("\n18H|Адрес на таблицата за пренасочване на адресите       |     |");
        printf("\n   |relocation table) в EXE файла                      |1 дума|%u",Buff.AdrTable);
        printf("\n1AH|Брой овърлейни процедури                           |1 дума|%u",Buff.BrOverlai);
        /*
        printf("\n  1CH\tБуферна памет        ??\t%u",Buff.
        printf("\n  ??H\tАдрес на таблицата на сегментните адреси за пренасочване  (relocation table)
          ??\t%u",Buff.
        printf("\n  ??H\tКод на програмата, данни и стеков сегмент     ??\t%u",Buff.
        */
        }
          fclose(in);
}


        // Извежда информация за файловете в дадена директория

        #include <stdio.h>
        #include <string.h>
        #include <dos.h>
        void   OutStuct(  struct  _find_t *c_file);
        void main(int argc, char *argv[], char *envp[])
        {
         struct _find_t  c_file;
         strcat(argv[1],"\\*.*");
          _dos_findfirst( argv[1],_A_RDONLY|_A_ARCH|_A_HIDDEN|_A_SYSTEM, &c_file );
          if (   (c_file.attrib &(_A_RDONLY|_A_ARCH|_A_HIDDEN|_A_SYSTEM)) ==
                  (_A_RDONLY|_A_ARCH|_A_HIDDEN|_A_SYSTEM)   )   {    OutStuct(  &c_file);     }
           while( _dos_findnext( &c_file ) == 0 )
           {
            if (  ( c_file.attrib &(_A_RDONLY|_A_ARCH|_A_HIDDEN|_A_SYSTEM)) ==
                  (_A_RDONLY|_A_ARCH|_A_HIDDEN|_A_SYSTEM) )  OutStuct(  &c_file);
           }
        }

        void   OutStuct(  struct  _find_t  *c_file)
        {
            int nDay =c_file->wr_date & 0x1f;
```

```
    int nMonth = (c_file->wr_date >> 5) & 0x0f;
    int nYear = (c_file->wr_date >> 9) +1980;
    int nSecond =c_file->wr_time & 0x1f;
    int nMinute = (c_file->wr_time >> 5) & 0x3f;
    int nHors    = (c_file->wr_time >> 11) & 0x1f;
    printf("\n%13s  %8lu",c_file->name,c_file->size);
    printf(" %02u.%02u.%04u",nDay,nMonth,nYear);
    printf(" %02u:%02u:%02u  ",nHors,nMinute,nSecond);
     if (   (c_file->attrib & _A_ARCH) )  printf("A");
     if (   (c_file->attrib & _A_HIDDEN))  printf("H");
     if (   (c_file->attrib & _A_NORMAL) )  printf("N");
     if (   (c_file->attrib & _A_RDONLY) )  printf("R");
     if (   (c_file->attrib & _A_SUBDIR) )  printf("D");
     if (   (c_file->attrib & _A_SYSTEM) )  printf("S");
     if (   (c_file->attrib & _A_VOLID) )  printf("S");
}
```

Управление на видео-системата в програмният език С.

Един от основните показатели на една програма е привлекателността на интерфейса между потребител и система който тя предлага. За да се създаде този интерфейс е необходимо да се познава видеосистемата и функциите предназначени за нейното управление. Съществуват множество видео-системи и видеорежими, като CGA,EGA, VGA, SVGA и др. Всеки един от тях предлага два основни режима на управление на дисплея, а именно текстов и графичен. В тази част ще се запознаем с основните функции предназначени за управление на видеосистемата.

Основната разликата между текстов и графичен режим е в управлението на видеопаметта. В текстов режим е достъпна памет в размер от 16К с началемюн адрес B800:0000. Тази памет е разделена на няколко видестраници в зависимост от зададения режим. За всеки един символ се заделят по два байта във видеопаметта. Първият байт съдържа ASCI кода на символа , а втория информация за цвета на символа и фона.

Пример:

```
void main()
{
  unsigned char __far *p = (unsigned char __far *)0xB8000000;
  for(int I=0; I<25*80*2;*(p+I)='*',*(p+I+1)=(char)(I%255),I+=2);
}
```

Този пример показва как може да се осъществи директен достъп до видеопаметта в текстов режим. В случая се запълва първата видеостраница в режим 80x25 със символа ‘*’, като за всяка позиция се задава различен цвят на фон и на символ.

При графичен режим управлението на видеопаменнта зависи от вида на видеоадаптера. За наше щастие фирмите производителки на С компилатори предодставят богат набор от функции предназначени за управление на видеосистемата в графичен и текстов режим.

Една програма работеща с видеосистемата трябва да има следната структура:

1. Включване на библиотеката GRAPH.H.
2. Определяне на видео режима и видео конфигурацията.
3. Запазване на старият видеорежим.
4. Задаване на желаният видеорежим.
5. Програмна част обезпечаваща потребителските изисквания за програмата.
6. Възстановяване на предишният видеорежим.

Графични функции

_displaycursor	_gettextcolor	_gettextcursor	attribute
_gettextposition	_gettextwindow	_outmem	_outtext
_scrolltextwindow	_settextcolor	_settextcursor	_settextposition
_settextwindow	_wrapon	_getcurrentposition	_getcurrentposition_w
_getphyscoord	_getviewcoord		
_getviewcoord_w	_getviewcoord_wxy	_setvieworg	_setviewport
_setwindow	_remapallpalette	_remappalette	_selectpalett
_getarcinfo	_getcolor	_getfillmask	_getlinestyle
_getwritemode	_setcolor	_setfillmask	_setlinestyle
_setwritemode	_arc	_arc_w	_arc_wxy
_ellipse,	_ellipse_w,	_ellipse_wxy	_floodfill
_floodfill_w	_getpixel	_getpixel_w	_lineto
_lineto_w	_moveto	_moveto_w	_pie
_pie_w	_pie_wxy	_polygon	_polygon_w
_polygon_wxy	_rectangle	_rectangle_w	_rectangle_wxy
_setpixel	_setpixel_w	_getimage	_getimage_w
_getimage_wxy	_imagesize	_imagesize_w	_imagesize_wxy
_putimage	_putimage_w	_getfontinfo	_getgtextextent
_getgtextvector	_outgtext	_registerfonts	_setfont
_setgtextvector	_unregisterfonts		

Определяне на конфигурацията.

Функция _getvideoconfig().

Една от първоначалните задачи в процеса на създаване от програмиста на собствени приложения, активно използващи възможностите на видеоадаптерите, се явява определянето на тип на видеоадаптера и видеодисплея, включени в компютъра. Най-простия начин да се реши тази задача е да се използва функцията _getvideoconfig().Функцията _getvideoconfig() запълва структурата videoconfig, определена във файла graph.h.

```
struct _videoconfig
{
        short numxpixels;    // Брой пиксели по оста X
        short numypixels;    // Брой пиксели по оста У
        short numtextcols;   // Брой на колоните в текстов режим
        short numtextrows;   // Брой на редовете в текстов режим
        short numcolors;     // Брой на цветовете
        short bitsperpixel;  // Number of bits per pixel
        short numvideopages; // Брой на свободните видео страници
        short mode;          // Текущ видеорежим
        short adapter;       // Активен видеоадаптер
        short monitor;       // Активен видеомонитор
        short memory;        // Размер на видеопаметта
};
```

Полето adapter определя типа на първоначалния адаптер.Той може да приема следните значения,определени във файла graph.h.

Константа	Наименование на видеоадаптера
_CGA	Color Graphics Adapter
_EGA	Enhanced Graphics Adapter
_HGC	Hercules Graphics Card
_MCGA	Multicolor Graphics Array
_MDPA	Monochrome Display Pr**int**er Adapter
_OCGA	Olivetti (AT&T) Color Graphics Adapter
_OEGA	Olivetti (AT&T) Enhanced Graphics Adapter
_OVGA	Olivetti (AT&T) Video Graphics Array
_VGA	Video Graphics Array
_SVGA	Super Video Graphics Array (VESA)

Полето monitor определя типа на използвания в дадения момент дисплей. то може да приема едно от следните значения определени във файла graph.h.

Константа	Значение
_ANALOG	Аналогов (монохромен и цветен)
_ANALOGCOLOR	Аналогов (цветен)
_ANALOGMONO	Монохромен аналогов
_COLOR	Цветен(или подобрен цветен, емулиращ цветен)
_ENHCOLOR	Подобрен цветен
_MONO	Монохромен

Примерна програма определяща режима на работа .

```
#include <stdio.h>
#include <graph.h>

void main()
{
        struct _videoconfig vc;
        _getvideoconfig(&vc); // запълване полетата на структурата
              // Извеждане на екрана на данните в структурата.
        printf("\n Тип на видеоадаптера: ");
        switch(vc.adapter)
        {
          case _MDPA:puts("MDA"); break;
          case _CGA:puts("CGA"); break;
          case _EGA:puts("EGA"); break;
          case _VGA:puts("VGA"); break;
          case _MCGA:puts("MCGA"); break;
          case _HGC:puts("Hercules"); break;
          default :puts("неизвестен");
        }
        printf("\n Тип на дисплея: ");
        switch(vc.monitor)
        {
          case _COLOR: puts("цветен или емулиращ го."); break;
```

```
        case _MONO: puts("монохромен."); break;
        case _ENHCOLOR: puts("подобрен цветен."); break;
        case _ANALOGMONO: puts("аналогов монохромен."); break;
        case _ANALOGCOLOR: puts("аналогов цветен."); break;
        case _ANALOG: puts("аналогов."); break;
        default: puts("неизвестен");
      }
      printf("\n Обем на видеопаметта %d Кбайта.",vc.memory);
      printf("\n Номер на режима %d .",vc.mode);
      printf("\n Брой допустими видео страници %d ",
                                    vc.numvideopages);
          // при графичен режим на работа
      if ( vc.numpixels){
      printf("\n Разрешаваща способност в пиксели %dx%d пиксела",
                          vc.numxpixels,vc.numypixels);
      printf("\n Количество битове на пиксел %d",vc.bitsperpixel);
      }
      printf("\n Брой символи на ред %d ",vc.numtextcols);
      printf("\n Брой редове %d ",vc.numtextrows);
      printf("\n Брой цветове %d ",vc.numcolors);
}
```

Друг способ на ниско ниво е използването на функциите от десето прекъсване 1Ah и 12h или проверка съдържанието на адресите 0000:0410 и 0000:0487.

Функции за управление на видеоадаптера.

Функция _displaycursor().

Функцията позволява да се гаси курсора в текстов режим.
Нейният прототип е:

short _displaycursor(**short** toggle)

Параметърът toggle е равен на _GCURSORON при разрешаване на курсора и _GCURSOROFF при забрана на курсора. Функцията връща предишното състояние на курсора.Описанието на функцията се намира във файла graph.h.

Пример за използването на функцията.

```
#include <stdio.h>
#include <graph.h>

void main()
{
  _clearscreen(_GCLEARSCREEN); // изчистване на екрана на дисплея
  _displaycursor(_GCURSOROFF); // ЗАГАСЯНЕ НА КУРСОРА
  printf("\n Курсорът е загасен");
  _getch();
_displaycursor(_GCURSORON); //възстановяване на курсора
  printf("\n Курсорът е включен");
  _getch();
}
```

Функция _gettextcursor()

Функцията позволява да се определи формата на курсора, т.е. положението на началната и крайната линия.

short _gettextcursor(**void**);

Старшият байт връща разположението на горната линия а младшият на долната. Функцията връща -1 при наличието на грешка например ако в момента е активен графичен режим.

Функция _settextcursor()

За установяване на формат на курсора се използва функцията _settextcursor() от стандартните функции на Microsoft C. Фактически функцията е реализирана чрез функция 01h на прекъсване 10h. Форматът на функцията е :

short _settextcursor(**short** share);

Параметърът share задава новата форма на курсора.Старшият байт определя горната граница а младшият долната. Функцията връща предишната стойност на курсора или -1 при грешка. Следващата програма демонстрира функциите _gettextcursor и _settextcursor.

```
#include <conio.h>
#include <stdio.h>
#include <graph.h>

/* Macro to define cursor lines */
#define CURSOR(top,bottom) (((top) << 8) | (bottom))

void main()
{
  short oldcursor, newcursor;
  unsigned char top, bottom;
  char buffer[80];
  struct _videoconfig vc;

  /* Съхранява предишният курсор */
  oldcursor = _gettextcursor();
  _clearscreen( _GCLEARSCREEN );
  _displaycursor( _GCURSORON );

  /* Промяна формата на курсора */
  for( top = 7, bottom = 7; top; top-- )
  {
    _settextposition( 1, 1 );
    sprintf( buffer, "Top line: %d   Bottom line: %d  ",top, bottom );
    _outtext( buffer );
    newcursor = CURSOR( top, bottom );
    _settextcursor( newcursor );
    _getch();
  }

  _outtext( "\nCursor off" );
  _displaycursor( _GCURSOROFF );
  _getch();
  _outtext( "\nCursor on" );
  _displaycursor( _GCURSORON );
  _getch();

  /* възтановява предишният  курсор. */
  _settextcursor( oldcursor );
  _clearscreen( _GCLEARSCREEN );
}
```

Функция _gettextposition()

Определя положението на курсора върху екрана.Същата операция може да се се извърши чрез функцията 03h от прекъсване 10h.

struct rccoord _gettextposition(**void**);

Функцията връща в структура rccoord текущите координати. Ще отбележим, че началото на координатната система е в горният ляв ъгъл на екрана и има координати(1,1) при това оста X е хоризонталната ос а оста У е вертикалната. Структурата rcoord е декларирана във файла graph.h и има следният вид:

```
struct _rccoord
 {
    short row;
    short col;
 };
```

Функция _settextposition()

Kakto и функция 02h от прекъсване 10h функцията _settextposition() от стандартните библиотеки на Microsoft C установява текущата позиция на курсора.

struct rccoord _settextposition(**short** row,**short** column);

Параметрите на функцията задават новите координати на курсора на екрана. Първият параметър задава редът, а втория колоната. Функцията връща структура от типа rccoord съдържаща предишните координати на курсора. Долният пример илюстрира използването на функциите _gettextposition() и _settextposition():

```
#include <conio.h>
#include <stdio.h>
#include <graph.h>
char buffer [80];
void main()
{
  short i,j;
  struct _rccoord oldpos;
  oldpos = _gettextposition();
  _clearscreen( _GCLEARSCREEN );
  for( i = 0; i<9; i++ )
  {
    for( j = 0; j<12; j++ )
    {
      _settextposition(2*j+1,9*i+ 1 );
      printf(".(%d,%d)",9*i+1,2*j+1);
    }
   _getch();
  _settextposition( oldpos.row, oldpos.col );
  _clearscreen( _GCLEARSCREEN );
}
```

Функция _getbkcolor()

Функцията позволява да се разбере текущият цвят на фона (бит d4-d6 в байта на атрибута).

long _getbkcolor(**void**);

В текстов режим функцията връща индекса на цвета, а в графичен универсалното значение на цвета.

номер	константа
0	_BLACK
1	_BLUE
2	_GREEN
3	_CYAN
4	_RED
5	_MAGENTA
6	_BROWN
7	_WHITE
8	_GRAY
9	_LIGHTBLUE
10	_LIGHTGREEN
11	_LIGHTCYAN
12	_LIGHTRED
13	_LIGHTMAGENTA
14	_YELLOW
15	_BRIGHTWHITE

Функция _setbkcolor()

Функцията задава текущия цвят на фона.

long _setbkcolor(**long** color);

В текстов режим параметъра трябва да съдържа индекса на установения цвят например ако искаме да установим бял цвят то трябва да зададем следния вид на функцията _setbkcolor(15L). При използване на функцията за промяна на фона цвета на символите си остава предишният.В графичен режим параметърът трябва да съдържа константа съответстваща на дадения цвят. Например за горният случай е необходим следният синтаксис на функцията _setbkcolor(_BRIGHTWHITE). Константите са определени във файла graph.h по следния начин:

```
#define _BLACK0x000000L
#define _BLUE   0x2a0000L
#define _GREEN0x002a00L
#define _CYAN   0x2a2a00L
#define _RED            0x00002aL
#define _MAGENTA            0x2a002aL
#define _BROWN              0x00152aL
#define _WHITE          0x2a2a2aL
#define _GRAY           0x151515L
#define _LIGHTBLUE   0x3F1515L
#define _LIGHTGREEN0x153f15L
```

```
#define _LIGHTCYAN  0x3f3f15L
#define _LIGHTRED           0x15153fL
#define _LIGHTMAGENTA   0x3f153fL
#define _YELLOW             0x153f3fL
#define _BRIGHTWHITE        0x3f3f3fL
```

Дадените константи представляват цвета във формат съответстващ на таблицата на цветовете на цифрово-аналоговия преобразувател на VGA адаптера. Всяка константа се състои от три байта, при което от всеки байта са значещи само шест бита. Трябва да подчертаем че при изменението на цвета на фона в текстов режим цвета на фона на вече въведените символи не се променя , а в графичен режим се променя незабавно. В текстов режим функцията връща стария индекс на цвета а в графичен универсалния универсалното значение на цвета на фона.

Функция _gettextcolor()

С помощта на тази функция може да се определи текущият цвят на символите.

```
short _gettextcolor();
```

Функцията връща индекса на цвета на символите.

Функция _settextcolor()

Функцията установява цвета на символите извеждани на екрана в текстов режим.

```
short _settextcolor(short index);
```

Параметърът съдържа индекса на новия цвят на символите.Ако към индекса се прибави 10h символите ще мигат. Функцията връща предишният индекс на цвета на символите.Ще покажем използването на функциите _getbkcolo, _setbkcolor, _gettextcolor и _settextcolor в следният пример:

```
#include <conio.h>
#include <stdio.h>
#include <graph.h>

char buffer [80];

void main()
{
  short blink, fgd, oldfgd;
  long bgd, oldbgd;
  struct _rccoord oldpos;

  /* Съхранява оригиналния цвят на фона и символите, както и
    текущата позиция. */
  oldfgd = _gettextcolor();
  oldbgd = _getbkcolor();
  oldpos = _gettextposition();
  _clearscreen( _GCLEARSCREEN );
  /* Първият и последният мигат. */
  for( blink = 0; blink <= 16; blink += 16 )
  {

    /* Цикъл за 8 цвята на фона. */
```

```
    for( bgd = 0; bgd < 8; bgd++ )
    {
       _setbkcolor( bgd );
       _settextposition((short)bgd + ((blink/16) * 9) + 3,1);
       _settextcolor( 15 );
       sprintf(buffer, "Back: %d Fore:", bgd );
       _outtext( buffer );

       /* Цикъл за 16 поредни цвята на текста. */
       for( fgd = 0; fgd < 16; fgd++ )
       {
          _settextcolor( fgd + blink );
          sprintf( buffer, " %2d ", fgd + blink );
          _outtext( buffer );
       }
    }
  }
  _getch();

  /* възстановява оригиналните цвят на фона и символите */
  _settextcolor( oldfgd );
  _setbkcolor( oldbgd );
  _clearscreen( _GCLEARSCREEN );
  _settextposition( oldpos.row, oldpos.col );
}
```

Функция _outtext()

Функцията извежда на екрана текст във всички режими на работа на видеоадаптера. Форматът на функцията е следният:

void _outtext(unsigned **char** _far *text);

Единственият параметър на функцията е указател към извеждания низ. Извеждането започва от текущата позиция, с текущия цвят на символите зададен с функцията _settextcolor();

Функции : _setactivepage(), _setvideopage(), _getactiwepage() и getvisualpage().

Функциите се използват за управление на видеоадаптера при работа с няколко видео страници. Те обезпечават възможността за подготовка на изображението на няколко страници и бързото им извеждане на екрана. Функцията _setactivepage() насочва изхода към дадена страница,а функцията _setvisualpage() задава видимата страница.Функциите за работа с видео страниците могат да се използват съвместно със следните функции _gettextcursor(), _settextcursor(), _displaycursor(), _settextcolor(), _gettextcolor(), _getbkcolor(), _setbkcolor(), _gettextposition(), _settextposition(), _outtext(), _settextwindow().

Функцията за управлението на страницата има следния синтаксис:

short _setactivepage(**short** page);

Параметърът на функцията задава активната видео страница, в която ще бъде насочен последващият изход. Ако изпълнението е успешно функцията връща номера на предишната активна страница. При грешка връща -1.Грешка може да възникне ако се зададе номер на несъществуваща видео страница.

Прототип на функцията _setvisualpage();

short _setvisualpage(**short** page);

Параметърът на функцията задава текущата изобразявана страница. Функцията връща номера на предишната активна страница или -1 при наличие на грешка. Възниква грешка в случаите когато видеоадаптера не поддържа видео страница с такъв номер. В този случай смяната на видео страниците не се извършва.

short _getactivepage(**void**);

Функцията дава номера на активната видео страница в момента. Неин прототип е и функцията _getvisualpage(), която връща номера на текущата видима страница.

short _getvisualpage(**void**);

Тук ще приведем една примерна програма използваща видео страниците.

```
#include <conio.h>
#include <graph.h>
#include <time.h>
#include <stdlib.h>

void delay( clock_t wait );        /* Prototype */
char *jumper[4][3] = { { { "\\o/" }, { " O "  }, { "/ \\" } },
                       { {  "_o_" }, { " O "  }, { "( )"  } },
                       { {  " o " }, { "/O\\" }, { "/ \\" } },
                      { {  " o " }, { " O "  }, { "( )"  } } };
void main()
{
  short oldvpage, oldapage, page, row, col, line;
  struct _videoconfig vc;
  _getvideoconfig( &vc );
  if( vc.numvideopages < 4 )
    exit( 1 );      // монохромен
  oldapage  = _getactivepage();
  oldvpage  = _getvisualpage();
  if( !_setvideomoderows( _TEXTBW40, 25 ) )
    exit( 1 );             /* режим  40 колони */
  _displaycursor( _GCURSOROFF );
  /* изобразява всички страници */
  for( page = 0; page < 4; page++ )
  {
    _setactivepage( page );
    for( row = 1; row < 23; row += 7 )
    {
      for( col = 1; col < 37; col += 7 )
      {
        for( line = 0; line < 3; line++ )
        {
          _settextposition( row + line, col );
          _outtext( jumper[page][line] );
        }
      }
    }
  }
  while( !_kbhit() )
    /* Циклично преминаване през страниците. */
    for( page = 0; page < 4; page++ )
    {
```

```
            _setvisualpage( page );
            delay( 100L );
        }
    _getch();
    /* Възстановява оригиналната страница. */
    _setvideomode( _DEFAULTMODE );
    _setactivepage( oldapage );
    _setvisualpage( oldvpage );
}

/* Пауза специфицирана от числото в микросекунди. */
void delay( clock_t wait )
{
    clock_t goal;
    goal = wait + clock();
    while( goal > clock() ) ;
}
```

Функция _setvideomode();

Това е една от най важните функции. Тя позволява да се измени режимът на работа на видеоадаптера. Тя има следния формат:

short _setvideomode(**short** mode);

Параметърът на функцията mode определя новият режим на видеоадаптера и може да бъде една от стойностите в таблицата.

Mode	Type	Size	Colors	Adapter
_DEFAULTMODE	Режим съществуващ при стартирането			
_MAXRESMODE	Максимален графичен режим			
_MAXCOLORMODE	Графичен режим с максимален брой цветове			
_TEXTBW40	BW/T	40 columns	32	CGA
_TEXTC40	C/T	40 columns	32	CGA
_TEXTBW80	BW/T	80 columns	32	CGA
_TEXTC80	C/T	80 columns	32	CGA
_MRES4COLOR	C/G	320 x 200	4	CGA
_MRESNOCOLOR	BW/G	320 x 200	4	CGA
_HRESBW	BW/G	640 x 200	2	CGA
_TEXTMONO	M/T	80 columns	32	MDPA
_HERCMONO *	M/G/H	720 x 348	2	HGC
_MRES16COLOR	C/G	320 x 200	16	EGA
_HRES16COLOR	C/G	640 x 200	16	EGA
_ERESNOCOLOR	M/G	640 x 350	4	EGA
_ERESCOLOR	C/G	640 x 350	16/4	EGA
_VRES2COLOR	C/G	640 x 480	2	VGA
_VRES16COLOR	C/G	640 x 480	16	VGA
_MRES256COLOR	C/G	320 x 200	256	VGA
_ORESCOLOR	C/G	640 x 400	1 of 16	OGA

Тези константи са дефинирани във файла graph.h. Стойностите на първите три константи зависят от инсталирания видеоадаптер.Функцията setvideomode() връща броя на текстовите страници в установения режим на работа на видеоадаптера. Ако е настъпила грешка, например изисквания режим не се поддържа от видеоадаптера, функцията връща

0.Режимът използващ константите _MAXRESMODE, _MAXCOLORMODE в зависимост от конфигурацията на видео системата използва един от следните режими:

Adapter/Monitor	_MAXRESMODE	_MAXCOLORMODE
MDPA	fail	fail
HGC	_HERCMONO	_HERCMONO
CGA color	_HRESBW	_MRES4COLOR
CGA noncolor	_HRESBW	_MRESNOCOLOR
OCGA	_ORESCOLOR	_MRES4COLOR
OEGA color	_ ORESCOLOR	_ERESCOLOR
EGA color 256k	_HRES16COLOR	_HRES16COLOR
EGA color 64k	_HRES16COLOR	_HRES16COLOR
EGA ecd 256k	_ERESCOLOR	_ERESCOLOR
EGA ecd 64k	_ERESCOLOR	_HRES16COLOR
EGA mono	_ERESNOCOLOR	_ERESNOCOLOR
MCGA	_VRES2COLOR	_MRES256COLOR
VGA	_VRES16COLOR	_MRES256COLOR
OVGA	_VRES16COLOR	_MRES256COLOR
SVGA	_VRES256COLOR *	_VRES256COLOR *

Режимите TEXTC40,TEXTBW40,TEXTC80,TEXTBW80 различават само използването на цветната палитра. В режимите TEXTBW40 и TEXTBW80 могат та използват само нюансите на сивото. Видеоадаптера Hercules може да работи в режим _HERCMODE. Този монохромен режим е с разрешаваща способност 720x348.Той обезпечава страница с размер 25x80 с размер на символите 9x14 пиксела.Преди да изпълните вашата програма в режим _HERCMONO трябва да установите специален драйвер на видеоадаптера Hercules - програмата msherc.com, която може да се стартира непосредствено преди програмата или да се стартира непосредствено от нея чрез функциите system или exec.При едновременното използване на Hercules и EGA адаптери трябва да се стартира програмата msherc.com с ключ /H. След това видеоадаптера ще използва само за Hercules само една от двете видео страници и няма да има конфликт между видеоадаптерите. Сега ще покажем един прост пример за използването на видеоадаптерите.

```
#include <conio.h>
#include <stdio.h>
#include <graph.h>

short modes[] = { _TEXTBW40,    _TEXTC40,     _TEXTBW80,
          _TEXTC80,     _MRES4COLOR,  _MRESNOCOLOR,
          _HRESBW,      _TEXTMONO,    _HERCMONO,
          _MRES16COLOR, _HRES16COLOR, _ERESNOCOLOR,
          _ERESCOLOR,   _VRES2COLOR,  _VRES16COLOR,
          _MRES256COLOR, _ORESCOLOR
         };
char *names[] = { "TEXTBW40",   "TEXTC40",    "TEXTBW80",
          "TEXTC80",    "MRES4COLOR", "MRESNOCOLOR",
          "HRESBW",     "TEXTMONO",   "HERCMONO",
          "MRES16COLOR", "HRES16COLOR", "ERESNOCOLOR",
          "ERESCOLOR",  "VRES2COLOR", "VRES16COLOR",
          "MRES256COLOR","ORESCOLOR"
         };
```

```
short rows[] = { 60, 50, 43, 30, 25 };  /* Възможен брой редове */

void main()
{
 short c, i, j, x, y, row,num = sizeof(modes)/sizeof(modes[0]);
 struct _videoconfig vc;
 char b[500];            /* Буфер за стринга  */

 _displaycursor( _GCURSOROFF );

 /* Опитва всички режими. */
 for( i = 0; i <= num; i++ )
 {
  for (j = 0; j < 5; j++ )
  {
     /* Опитва възможния брой редове. */
     row = _setvideomoderows( modes[i], rows[j] );
     if( (!row) || (rows[j] != row) )
        continue;
     else
     {
           _getvideoconfig( &vc );
             y = (vc.numtextrows - 12) / 2;
             x = (vc.numtextcols - 25) / 2;
  _settextwindow( y, x, vc.numtextrows - y, vc.numtextcols - x );
  c  = sprintf( b,     "Video mode: %s\n", names[i] );
  c += sprintf( b + c, "X pixels:   %d\n", vc.numxpixels );
  c += sprintf( b + c, "Y pixels:   %d\n", vc.numypixels );
  c += sprintf( b + c, "Columns:    %d\n", vc.numtextcols );
  c += sprintf( b + c, "Rows:       %d\n", vc.numtextrows );
  c += sprintf( b + c, "Colors:     %d\n", vc.numcolors );
  c += sprintf( b + c, "Bits/pixel: %d\n", vc.bitsperpixel );
  c += sprintf( b + c, "Pages:      %d\n", vc.numvideopages );
  c += sprintf( b + c, "Mode:       %d\n", vc.mode );
  c += sprintf( b + c, "Adapter:    %d\n", vc.adapter );
  c += sprintf( b + c, "Monitor:    %d\n", vc.monitor );
  c += sprintf( b + c, "Memory:     %d",   vc.memory );
  _outtext( b );
   _getch();
           }
       }
   }
   _displaycursor( _GCURSORON );
   _setvideomode( _DEFAULTMODE );
}
```

Функция _clearscreen().

Функцията изчиства екрана или отделна област от него.Изчистваната област се запълва с текущият цвят на фона. Функцията има следният прототип:

void _clearscreen(**short** area);

Параметърът area може да приема една от следните стойности:

_GCLEARSCREEN	изчиства целия екран
_GWINDOW	изчиства текущия текстов прозорец
_GWINPORT	текущата логическа система от координати

Функция _settextwindow()

Функцията задава текстов екран в който ще се извежда информацията Извеждането на информацията в прозореца се извършва от горе на долу.След запълването на прозореца, автоматично се извършва скрол на текста. Функцията _settextwindow() не действа на изпълнението на функцията _ottext() (за това трябва да се изпълни функцията _setviewport()). Синтаксисът на функцията е следният:

void _settextvindow(**short** y_up,**short** x_left, **short** _down, **short** x_right);

Параметрите (y_up,x_left) определят горния ляв ъгъл на прозореца, а (y_down,x_right) определят долния десен ъгъл на екрана.

Функция _scrolltextwindow()

Функцията _scrolltextwindow() извършва скрол на текста в прозорец а на зададен брой линии.Тя има следния синтаксис:

void _scrolltextwindow(**short** lines);

Параметъра lines задава броя на преместваните линии. Ако параметърът е положителен то скролът се извършва нагоре, ако е отрицателен се извършва надолу.

Функция _gettextwindow()

Тази функция връща координатите на текущия прозорец. Тя има следния синтаксис:

void _gettextwindow(**short** __far *r1, **short** __far *c1, **short** __far *r2, **short** __far *c2);

Следващият пример илюстрира действието на тези функции.

```
#include <stdio.h>
#include <conio.h>
#include <graph.h>

void deleteline( void );
void insertline( void );
void status( char *msg );

void main()
{
      short row;
      char  buf[40];

 /* Установява размера на прозореца за скрол ; извежда текст */
      _settextrows( 25 );
      _clearscreen( _GCLEARSCREEN );
      for( row = 1; row <= 25; row++ )
      {
        _settextposition( row, 1 );
   sprintf( buf, "Line %c           %2d", row + 'A' - 1, row );
        _outtext( buf );
      }
      _getch();
      _settextwindow( 1, 1, 25, 10 );
      /* Изтрива няколко линии. */
      _settextposition( 11, 1 );
```

```
        for( row = 12; row < 20; row++ )
          deleteline();
        status( "Deleted 8 lines" );
        /* Вмъква няколко линии. */
        _settextposition( 5, 1 );
        for( row = 1; row < 6; row++ )
          insertline();
        status( "Inserted 5 lines" );
        /* Скрол нагоре и надолу. */
        _scrolltextwindow( -7 );
        status( "Scrolled down 7 lines" );
        _scrolltextwindow( 5 );
        status( "Scrolled up 5 lines" );
        _setvideomode( _DEFAULTMODE );
}

/* Изтрива линии и прави скрол на горе в текущия текстов прозорец
 * Записва и възстановява размера на оригиналния прозорец. */
void deleteline()
{
        short left, top, right, bottom;
        struct _rccoord rc;
        _gettextwindow( &top, &left, &bottom, &right );
        rc = _gettextposition();
        _settextwindow( rc.row, left, bottom, right );
        _scrolltextwindow( _GSCROLLUP );
        _settextwindow( top, left, bottom, right );
        _settextposition( rc.row, rc.col );
}

/* Вмъква няколко празни линии и прави скрол на горе на текущия текстов прозорец. Записва и
възстановява размерите на оригиналния прозорец */
void insertline()
{
        short left, top, right, bottom;
        struct _rccoord rc;
        _gettextwindow( &top, &left, &bottom, &right );
        rc = _gettextposition();
        _settextwindow( rc.row, left, bottom, right );
        _scrolltextwindow( _GSCROLLDOWN );
        _settextwindow( top, left, bottom, right );
        _settextposition( rc.row, rc.col );
}

/* Извежда и изтрива собствения си статус. */
void status( char *msg )
{
        short left, top, right, bottom;

        _gettextwindow( &top, &left, &bottom, &right );
        _settextwindow( 1, 50, 2, 80 );
        _outtext( msg );
        _getch();
        _clearscreen( _GWINDOW );
        _settextwindow( top, left, bottom, right );
}
```

Основни графични функции.

Тези функции изобразяват основните графични обекти, такива като точка, правоъгълник, елипса, сектор , както и съхраняване на правоъгълна област от екрана.

Функция _setpixel().

Функцията задава цвета на пиксела в зададената позиция на екрана. Функцията има следния синтаксис:

short _setpixel(**short** x,**short** y);

Цвета на пиксела се задава предварително с функцията _setcolor(). Координатите на пиксела се задават чрез параметрите x и y.Функцията връща предишния цвят на пиксела или -1 при грешка.

Функция _lineto().

Функцията изчертава на екрана линия от текущата позиция до позицията зададена в параметрите й. Функцията има следния синтаксис.

short _lineto(**short** x, **short** y);
short _lineto_w(**double** wx, **double** wy);

След изпълнението на функцията текущата координата става координатата на точката зададена като параметър на функцията. Линията се изчертава с текущия цвят зададен с _setcolor() и текущия стил зададен, чрез _setstyle(). Функцията връща ненулева стойност при успешно изпълнение или 0 при грешка.

Функция _moveto()

Тази функция задава текущата координата.Използва се съвместно с _lineto() за изобразяване на линия. Прототипът на функцията _moveto() има следния вид:

struct xycoord_moveto(**short** x,**short** y);

Новите координати на текущата точка съответстват на параметрите на функцията (точка с координати (x,y)). Функцията връща в структурата xycoord координатите на предната. Структурата xycoord,описана във файла graph.h има следния вид:

```
struct _xycoord
{
   short xcoord;
   short ycoord;
};
```

Функция _rectangle()

Функцията рисува правоъгълник.В зависимост от параметрите на функцията правоъгълникът може да бъде запълнен или незапълнен. Прототипът на функцията има следния вид :

short _rectangle(**short** control, **short** x1, **short** y1, **short** x2, **short** y2);
short _rectangle_w(**short** control, **double** wx1, **double** wy1, **double** wx2, **double** wy2);
short _rectangle_wxy(**short** control,**struct** _wxycoord __far *pwxy1,

struct _wxycoord __far *pwxy2);

control: _GBORDER, _GFILL**INT**ERIOR

Правоъгълникът се определя от координатите на двойката срещуположни ъгли (x1,y1) и (x2,y2). Границите на правоъгълника се изчертават с текущия цвят и текущия стил на линията.Ако параметъра control е _GFILLE**INT**ERIOR, то правоъгълника се запълва, ако е _GBORDER, той не се запълва. Функцията връща ненулева стойност при правилно изпълнение или 0 при грешка.

Функция _ellipse()

Функцията рисува елипса. Синтаксисът на функцията е следния:

short _ellipse(**short** control, **short** x1, **short** y1, **short** x2, **short** y2);
short _ellipse_w(**short** control,**double** wx1,**double** wy1,**double** wx2,**double** wy2);
short _ellipse_wxy(**short** control,**struct** _wxycoord __far *pwxy1, **struct** _wxycoord __far *pwxy2)
control: _GFILL**INT**ERIOR, _GBORDER

Елипсата се задава чрез правоъгълника в, който се вписва. Координатите се задават, чрез (x1 , y1) горен ляв ъгъл и (x2,y2) долен десен ъгъл. В променливата control се задава типът на запълване на фигурата.При _GFILLE**INT**ERIOR се запълва, а при _GBORDER не се запълва. За изменение на текущия цвят се използва функцията _setcolor(). Ako аргументите на функцията определят да се изчертае вертикална или хоризонтална линия или точка тя не се изчертава. Функцията връща ненулева стойност при успех и нула при грешка.

Функция _arc().

Функцията чертае дъга на елипса. Синтаксисът и е :

short _arc(**short** x1, **short** y1, **short** x2, **short** y2, **short** x3, **short** y3, **short** x4, **short** y4);
short _arc_w(**double** x1,**double** y1,**double** x2, **double** y2,**double** x3, **double** y3,**double** x4,**double** y4)
short _arc_wxy(**struct** _wxycoord __far *pwxy1, **struct** _wxycoord __far *pwxy2,
struct _wxycoord __far *pwxy3, **struct** _wxycoord __far *pwxy4);

Елипсата се задава с правоъгълника в който тя се вписва. Координатите на правоъгълника се задават чрез координатните двойки (x1,y1) и (x2,y2). Дъгата се построява от точка на пресичане на елипсата с линията свързваща центъра на елипсата с точката зададена с координати (x3 , y3) до точката на пресичане на елипсата с линията свързваща центъра на елипсата и точката с координати (x4,y4). Изчертаването става в посока обратна на часовниковата стрелка. Функцията връща не нулева стойност при успех и 0 при грешка.

Функция _pie().

Функцията чертае сектор от елипса. Синтаксисът и е:

short _pie(**short** control,**short** x1, **short** y1,**short** x2, **short** y2,**short** x3, **short** y3, **short** x4, **short** y4)
short _pie_w(**short** control, **double** x1, **double** y1, **double** x2, **double** y2,
double x3, **double** y3, **double** x4, **double** y4);
short _pie_wxy(**short** control, **struct** _wxycoord __far *pwxy1, **struct** _wxycoord __far *pwxy2,
struct _wxycoord __far *pwxy3, **struct** _wxycoord __far *pwxy4);

control: _GFILL**INT**ERIOR, _GBORDER

Елипсата се задава с правоъгълника в който тя се вписва. Координатите на правоъгълника се задават чрез координатните двойки (x1,y1) и (x2,y2). Секторът се построява от точка на пресичане на елипсата с линията свързваща центъра на елипсата с точката зададена с координати (x3,y3) до точката на пресичане на елипсата с линията свързваща центъра на елипсата и точката с координати(x4,y4). Изчертаването става в посока обратна на часовниковата стрелка. Функцията връща ненулева стойност при успех и 0 при грешка. Следващата примерна програма демонстрира действието на функциите.

```
#include <conio.h>
#include <stdlib.h>
#include <graph.h>

void main()
{
  short x, y;
  struct _xycoord xystart, xyend, xyfill;
  if( !_setvideomode( _MAXRESMODE ) )  /* Задава валидния графичен режим */
        exit( 1 );        /* Грешка  */
   for( x = 10, y = 50; y < 90; x += 2, y += 3 )/* Чертае пиксел */
        _setpixel( x, y );
      _getch();
      for( x = 60, y = 50; y < 90; y += 3 ) /* Чертае линия */
      {
        _moveto( x, y );
        _lineto( x + 20, y );
      }
      _getch();
      x = 110; y = 70;            /* Чертае квадрат */
      _rectangle( _GBORDER,       x - 20, y - 20, x, y );
      _rectangle( _GFILL**INT**ERIOR, x + 20, y + 20, x, y );
      _getch();
      x = 160;                    /* Чертае елипса     */
      _ellipse( _GBORDER,       x - 20, y - 20, x, y );
      _ellipse( _GFILL**INT**ERIOR, x + 20, y + 20, x, y );
      _getch();
      x = 210;                /* Чертае област */
      _pie(_GBORDER,x-20,y-20,x,y,x-10,y-20,x-20,y-10);
      _pie(_GFILL**INT**ERIOR,x+20,y+20,x,y,x,y+10,x+10,y);
      x=260;/*Чертаедъга*/
      _arc(x-20,y-20,x,y,x-10,y-20,x-20,y-10);
      _arc(x+20,y+20,x,y,x,y+10,x+10,y);
      _getarcinfo( &xystart, &xyend, &xyfill );
      _moveto( xystart.xcoord, xystart.ycoord );
```

```
        _lineto( xyend.xcoord,   xyend.ycoord );
        _floodfill( xyfill.xcoord, xyfill.ycoord, _getcolor() );
        _getch();
        _setvideomode( _DEFAULTMODE );
    }
```

Функция _setcolor()

В графичен режим функцията задава цвета използван от функциите изчертаващи примитивите. Синтаксисът и е :

short _setcolor(**short** color);

Параметърът color трябва да съдържа индексът на цвета. Функцията връща номера не предишния цвят или -1 при грешка.

Функция _selectpalette()

Функцията изменя цвета на палитрата при режим _MRES4COLOR и _MRESNOCOLOR. Тези режими съдържат четири цвята, цветът на фона се избира отделно и три цвята избирани от функцията _selectpalette(). Функцията има следния синтаксис.

short _selectpalette(**short** number);

Единственият параметър на функцията има различни значения за различните режими на видеоадаптера.В режим _MRES4COLOR параметъра number избира една от следните четири възможни палитри.

Палитри:

Number	Color 1	Color 2	Color 3
0	Green	Red	Brown
1	Cyan	Magenta	White
2	Light green	Light red	Yellow
3	Light cyan	Light magenta	Bright white

В режим _MRESNOCOLOR за видеоадаптер CGA с цветен дисплей се използват следните цветови палитри.

Палитра:

Number	Color1	Color2	Color3
0	Blue	Red	White
1	Blue	Red	White
2	Light Blue	Light Red	Bright White
3	Light Blue	Light Red	Bright White

Ако включен EGA адаптер то палитрите ще са:

Палитра Number	Color1	Color2	Color3

0	Green	Red	Brown
1	Cyan	Magenta	White
2	Light Green	Light Red	Yellow
3	Cyan	Magenta	White

При режим _ORESCOLOR цветовете ще са :

Index	Color	Index	Color
0	Black	8	Dark Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Brown	14	Yellow
7	White	15	Bright White

При правилно изпълнение функцията ще върне номера на предишната палитра.

Функция _setlinestyle()

Функцията определя маската използвана при чертането на линия. Синтаксисът и е :

void _setlinestyle(unsigned **short** mask);

Единственият параметър на функцията се явява набор от 16 бита. Всеки бит отговаря на пиксел от изобразяваната линия.При това ако бита на параметъра е 1 той ще се изобрази на екрана с текущият цвят. Ако битът е равен на 0 той не се изобразява.По премълчаване маската е равна на 0xFFFF.

Функция _setviewport()

Ние можем да използваме две различни координатни системи физическа и логическа. Началото на физическата координатна система е от горният десен ъгъл. Логическата координатна система задава своя собствена правоъгълна система. По премълчаване логическата и физическата координатна система съвпадат. За тяхното изменение се използват функциите _setviewport() и _setvieworg(). Функцията _setviewport() има следния синтаксис:

void _setviewport(**short** x1, **short** y1,**short** x2, **short** y2);

Нейните параметри задават правоъгълната област на екрана в който в последствие се отправя изхода. При това горният ляв ъгъл става начало на координатната система.

Функция _setvieworg()

Функцията премества началото на логическата координатната система от точката (0,0) в новата зададена точка. Синтаксисът и е :

struct _xycoord _setvieworg(**short** x, **short** y);

Функцията връща предишните координати.

Функция _setvideomoderows()

Тази функция позволява да се измени режима на работа на видеоадаптера В текстов режим позволява да се зададе нестандартен брой на линиите при извеждане на екрана. Синтаксисът и е :

short _setvideomoderows(**short** mode, **short** rows);

Първият параметър mode определя режима ,в който се превключва видеоадаптера. Този параметър може да приеме една от следните константи.

_DEFAULTMODE	_MAXRESMODE	_MAXCOLORMODE	_TEXTBW40
_TEXTC40	_TEXTBW80	_TEXTC80	_MRES4COLOR
_MRESNOCOLOR	_HRESBW	_TEXTMONO	_HERCMONO
_MRES16COLOR	_HRES16COLOR	_ERESNOCOLOR	_ERESCOLOR
_VRES2COLOR	_VRES16COLOR	_MRES256COLOR	_ORESCOLOR

Mode	Type	Size	Colors	Adapter
_DEFAULTMODE	Mode existing at startup			
_MAXRESMODE	Highest resolution in graphics mode			
_MAXCOLORMODE	Maximum colors in graphics mode			
_TEXTBW40	BW/T	40 columns	32	CGA
_TEXTC40	C/T	40 columns	32	CGA
_TEXTBW80	BW/T	80 columns	32	CGA
_TEXTC80	C/T	80 columns	32	CGA
_MRES4COLOR	C/G	320 x 200	4	CGA
_MRESNOCOLOR	BW/G	320 x 200	4	CGA
_HRESBW	BW/G	640 x 200	2	CGA
_TEXTMONO	M/T	80 columns	32	MDPA
_HERCMONO *	M/G/H	720 x 348	2	HGC
_MRES16COLOR	C/G	320 x 200	16	EGA
_HRES16COLOR	C/G	640 x 200	16	EGA
_ERESNOCOLOR	M/G	640 x 350	4	EGA
_ERESCOLOR	C/G	640 x 350	16/4	EGA
_VRES2COLOR	C/G	640 x 480	2	VGA
_VRES16COLOR	C/G	640 x 480	16	VGA
_MRES256COLOR	C/G	320 x 200	256	VGA
_ORESCOLOR	C/G	640 x 400	1 of 16	OGA

При VESA.

Mode	VESA	No.	Type1	Size	Colors	Adapter
_ORES256COLOR		0x0100	C/G	640x400	256	SVGA
_VRES256COLOR		0x0101	C/G	640x480	256	SVGA
_SRES16COLOR	2	0x0102	C/G	800x600	16	SVGA
_SRES256COLOR	2	0x0103	C/G	800x600	256	SVGA
_XRES16COLOR	3	0x0104	C/G	1024x768	16	SVGA
_XRES256COLOR	3	0x0105	C/G	1024x768	256	SVGA
_ZRES16COLOR	4	0x0106	C/G	1280x1024	16	SVGA
_ZRES256COLOR	4	0x0107	C/G	1280x1024	256	SVGA

Ако е установен текстовия режим то втория параметър rows задава броя на линиите. Вторият параметър може да бъде константата _MAXTEXTROWS. При този случай се установяват максималните за дадения видеоадаптер редове.

VGA 50
EGA 43
CGA 25.
Функцията връща броят на редовете на екрана или 0 при грешка.

Функция _getimage()

Функцията записва правоъгълна област от екрана в паметта. Синтаксисът и е :

```
void _getimage(short x1,short y1,short x2,short y2,char __huge *image)
void _getimage_w( double wx1, double wy1, double wx2, double wy2,char __huge *image );
void _getimage_wxy( struct _wxycoord __far *pwxy1, struct _wxycoord __far *pwxy2,
                    char __huge *image );
```

Координатите на областта се задават в параметрите (x1,y1) и (x2,y2), а буферът в който ще се записват image. Размерът на буфера може да се намери чрез функцията _imagesize().

Функция _putimage()

Функцията извежда на екрана пиктограмата записана в буфера image. При това първата точка съвпада с координатите (x,y). Параметъра action определя способа на записване на пиктограмата. Тези способи са:
. _

GAND	Пиктограмата се наслагва на екрана с логическо И.
_GOR	Пиктограмата се наслагва на екрана с логическо ИЛИ.
_GPRESET	Копиране на екрана като всеки цвят от пиктограмата се инвертира
_GPSET	Копиране на екрана като всеки цвят от пиктограмата се запазва
_GXOR	Пиктограмата се наслагва на екрана с логическо изключващо ИЛИ

Синтаксис:

void _putimage(**short** x, **short** y,**char** __huge *image, **short** action);
void _putimage_w(**double** wx, **double** wy,**char** __huge *image,**short** action)

Функция _imagesize()

Функцията връща размера на областта от екрана в байтове, необходими за съхраняването им в паметта. Задават се с две диагонални точки. Синтаксисът на функцията е:

long _imagesize(**short** x1, **short** y1, **short** x2, **short** y2);
long _imagesize_w(**double** wx1, **double** wy1,**double** wx2, **double** wy2);
long _imagesize_wxy(**struct** _wxycoord __far *pwxy1, **struct** _wxycoord __far *pwxy2);

Функцията определя размера на буфера по следния алгоритъм:

width= abs(x2) +1;
height = abs(y2) +1;
size = ((**long**)((width*bits_per_pixel+7)/8)*(**long**)height)+4;

Величината bits_per_pixel съдържа броят на битовете от видеопаметта определящи един пиксел. Този параметър на видео режима може да се получи от функцията _getvideoconfig(). Следващият пример ще илюстрира действието на тези функции.

```
#include <conio.h>
#include <stddef.h>
#include <stdlib.h>
#include <malloc.h>
#include <graph.h>

short action[5]  = { _GPSET,   _GPRESET, _GXOR,    _GOR, _GAND};
char *descrip[5] = {"PSET  ","PRESET","XOR   ","OR    ","AND   "};

void exitfree( char __huge *buffer );

void main()
{
  char  __huge *buffer;
  long  imsize;
  short i, x, y = 30;
  if( !_setvideomode( _MAXRESMODE ) )
    exit( 1 );
  imsize = _imagesize( -16, -16, +16, +16 );
  buffer = _halloc( imsize, 1 );
  if( buffer == NULL )
    exit( 1 );
  _setcolor( 3 );
  for ( i = 0; i < 5; i++ )
  {
    x = 50; y += 40;
    _ellipse( _GFILLINTERIOR, x - 15, y - 15, x + 15, y + 15);
    _getimage( x - 16, y - 16, x + 16, y + 16, buffer );
    if( _grstatus() )
```

```
        exitfree( buffer );      // наличие аа грешка
      _settextposition( 1, 1 );
      _outtext( descrip[i] );
      while( x < 260 )
      {
       x += 5;
       _putimage( x - 16, y - 16, buffer, action[i] );
       if( _grstatus() < 0 )
            exitfree( buffer );
      }
      _getch();
   }
   exitfree( buffer );
}

void exitfree( char __huge *buffer )
{
   _hfree( buffer );
   exit( !_setvideomode( _DEFAULTMODE ) );

}


//  Копира  текста от последните два реда на   първа видеостраница във файл. Видео режима е 40 x 25.

#include <stdio.h>
#include <graph.h>
#include <stdlib.h>
void main()
{
  int i;
  FILE *fp ;
  char Buff[80];
  char __far *p;
  p = (char __far *)0xB8000000;
  _setvideomode(_TEXTC40);
  for(i=0;i<40*25*2;i++)  printf("%1u",i%10); // Запълва последните два реда с цифри
  if (  ( fp=fopen("VIDEO.DAT","wb")) != NULL)
  {
     p = p + 22*40*2; // Начало на реда
     for( i = 0 ; i<80;Buff[i] = *p,p+=, i++) ; // Копиране на видеопаметта в буфер
     fwrite(Buff,80,1,fp); // Запис на буфера в паметта
     fclose(fp);
   }
  _setvideomode(_DEFAULTMODE);
}

// Изчертава на екрана елипса.
// Изчертаната елипса се запише във файл.

#include <stdio.h>
#include <string.h>
#include <graph.h>
#include <malloc.h>
#include <dos.h>
void main(int argc, char *argv[], char *envp[])
{
  FILE *fp ;
  char __huge *Buff;
  long Size;
```

```
 _setvideomode(_VRES16COLOR);
  _setcolor(15);
  strcpy(strchr(argv[0],'.'),".dat");
 if (  ( fp=fopen(argv[0],"wb")) != NULL)
  {
       _ellipse(_GFILLINTERIOR,50,50,100,200);
       Size = _imagesize(50,50,100,200);
       Buff = (char __huge  *)_halloc((int)Size,1);
       _getimage (  50,50,100,200,Buff);
       fwrite(Buff,(size_t)Size,1,fp);
       fclose(fp);
       _hfree(Buff);
  }
  getchar();
 _setvideomode(_DEFAULTMODE);
}

// Изчертава sin,cos,tg, arctg. Фигурите се изчертават от една
// функция, като и се предава като аргумент указател към функцията.

#include <stdio.h>
#include <graph.h>
#include <math.h>
void Disp(double (*Fun)(double));
void main()
{
  _setvideomode(_SRES16COLOR);
     Disp(sin);
     getchar();
     Disp(cos);
     getchar();
     Disp(tan);
     getchar();
     Disp(atan);
     getchar();
   _setvideomode(_DEFAULTMODE);
}
void Disp(double (*Fun)(double))
{
  double w=0,f,t=0;
  _clearscreen(_GCLEARSCREEN);
  _moveto_w  (1,300);
  _lineto_w(800,300);
  _moveto_w  (1,1);
  _lineto_w(1,600);
  _moveto_w(1,300);
  for(w=0;w<6.29;w+= 6.29/800)
  {
   f = 300 + 50*(Fun)(w);
   _lineto_w(t,f);
   t++;
  }
}
```

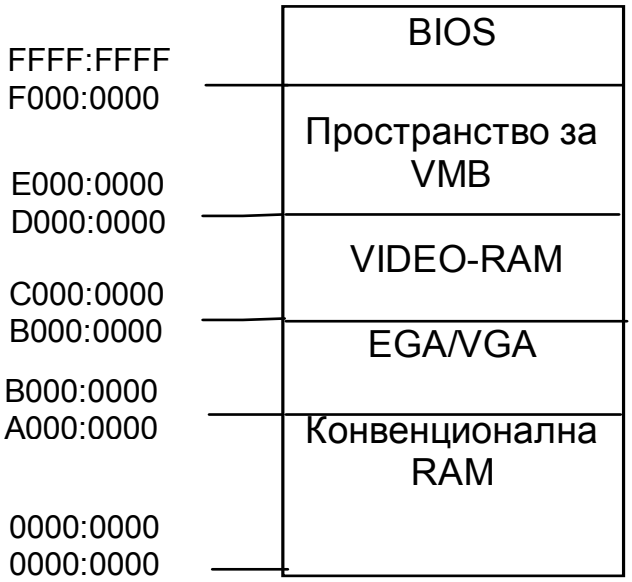
Управление на паметта

1.Общи сведения за разпределението на паметта.

Когато IBM PC бе разработен през 1980г., неговите възможности бяха много напредничави за времето си. Това бе вярно и за размера на неговата оперативна памет. Максимумът от 640К изглеждаше толкова много тогава, че никой не можеше да допусне какво ще прави потребителя с толкова много памет. Така например първите компютри бяха оборудвани с 64К , след това с 128К, по-късно с 256К памет. Минималният обем оперативна памет нарасна до 640К.

След като навлязохме в ерата на микропроцесорите 80486, графичните потребителски интерфейси и многозадачните операционни системи 640К не е достатъчна за пълното използване на пълните възможности на персоналните системи. Но е достигнат лимитът от максимално допустима адресируема памет, която може да се прескочи само с добавяне на чипове памет. Това е лимитът от 1М,установен от размера на адресируемото пространство на 8086.

Друг важен фактор засягащ пряко паметта е съвместимостта. За да се осъществи програмна съвместимост в машините, вариращи от най-простите XT до напълно оборудваните 486 системи процесорите трябва да бъдат съвместими и разпределението на паметта трябва да изпълнява определени стандарти. Следващата таблица показва базисната конфигурация на паметта на персоналните компютри:



фиг. 9.1

Както показва илюстрацията само първите 640К могат да бъдат използвани за RAM. Останалата памет над тази точка е запазена за видео RAM, хардуерни разширения и BIOS.

Паметта може да се разшири над един мегабайт. Днес повечето системи имат два и повече мегабайта памет, но тази памет не може да бъде адресирана от DOS. Понеже DOS работи в реален режим то максимално адресируемата памет от него е 1М. Това неудобство се преодолява, като се използват различни видове драйвери за управление на допълнителното количество памет. В зависимост от начина на достъп до тази памет тя се разделя на разширена (expandet) и допълнителна (extendet).

Разширената памет (Expandet memory) е допълнителна памет, която е разработена за PC/XT компютрите.Понеже процесорът е 8088 то тези компютри са ограничени да работят в рамките до 640К RAM. Чрез разширената памет се осъществява достъп до адресното пространство над 640К.

Допълнителната памет (Extendet memory) е памет, която е разработена за системи с процесор 80286 или по висок. Достъпът до нея става директно.

Паметта в границите от 640K до 1M е прието да се нарича висока памет (UMB). Тази област се използва от ROM BIOS, видеоадаптери, входно-изходната система, за достъп до разширената памет, както и за зареждане на операционната система и някой резидентни програми и драйвери.

2. Висока памет

Това е паметта между 640 KB и 1MB. В тази област от паметта са разположени входно-изходните портове, видеопаметта, ROM - BIOS, и 180KB RAM, която се използва за зареждане на резидентни програми посредством командата на DOS Lh, или за прозорец използван при достъпа до разширената памет.

3. Разширена памет.

PC или PC/XT машини са лимитирани до 640K конвенционална памет. Компютрите базирани на базата на процесори 80286 и по високи могат да адресират памет над 16M. Само че тези 16M са достъпни само когато компютъра работи в защитен режим. DOS работи само в реален режим,затова паметта над 1M е недостъпна за DOS приложенията.Преди известно време бе разработена система за достъп на DOS програми до паметта над 1M. Новоустановеният стандарт е наречен на името на фирмите, които го създават Lotus, IBM, Microsoft LIM. Този стандарт позволява до 8M да бъдат добавяни към персоналния компютър. Само 64K от тези 8M се намират в границите на 1M памет адресирана от микропроцесора 8086 в прозорец, който е наречен "Рамка за страница Page frame". Паметта инсталирана по този начин е наречена РАЗШИРЕНА
ПАМЕТ. Тази памет не бива да се бърка с допълнителната памет, която продължава в границите над 1M при PS/AT системите. Цялата система се нарича Система за разширена памет или EMS (Expandet Memory System).

LIM стандарта използва ПРЕВКЛЮЧВАНЕ НА БАНКИТЕ за достъп до паметта. Превключването на банките създава малък прозорец в паметта, чрез който става достъпа до паметта с адреси над адресното пространство на компютъра. Капацитетът на адресното пространство може да бъде адресирано до няколко мегабайта. Софтуера работи с хардуера, като премества част от тази памет в прозореца.Останалата памет е невидима за програмата.

Създаване на прозорец за паметта.

Понеже 64K от пространството от 1M не се използват конвенционалната памет,BIOS видеоадаптерите или останалите системни разширения е прието тя да се използва за достъп до разширената памет. Обикновено този прозорец се намира в сегмента с адрес D000H, но EMS хардуера позволява възможността този прозорец да бъде преместван.
Понеже прозорецът е в границите под 1M до него може да се осъществи достъп чрез обикновени асемблерски инструкции, подобни на тези за четене и запис във видеопаметта.

Странициране на паметта.

Рамката за странициране допълнително е разделена на страници от 16K. Това позволява на програмата да има достъп до четири различни страници от разширената памет. Регистрите на EMS картата позволяват на програмиста да установи коя страница от разширената памет се пренасочва в рамката на страницата, за да се осъществи достъп до нея от 8086. Този процес се нарича превключване на страниците.

Освен хардуерен EMS включва и софтуерен интерфейс, който управлява програмирането на регистрите на EMS и други задачи по управление на паметта. Този софтуерен интерфейс наречен Менажер на Разширената памет или EMM (Expandet Memary Manager), осигурява стандартния интерфейс, който осигурява достъп до EMS картите на различните производители. Това също се отнася и до разширения EMS стандарт (Extendet EMS или EEMS) разработен от AST Research, Quadram и Ashton-Tate, който надвишава LIM стандарта. Представители на този стандарт са : 386-To-The-Max(Qualitas), QEMM-386(Quarterdeck), Microsoft Windows. Горните продукти са базирани на виртуалния 8086 режим на работа на процесора 80386.Този режим позволява да се премества памет извън рамките на 1M и да се оперира с нея,като с нормална страница от паметта. В сравнение с конвенционалния хардуер изпъкват следните предимства:

- Без емулация на EMS до тази памет се осъществява достъп като до допълнителна памет.
- Понеже е хардуерно входно-изходното адресиране достъпът е по бърз.
- По ниска цена.
- Освобождава се един слот за разширяване.

EMS има следните възможности(версия 4.0):

- Поддържа памет от 32M.
- Възможност за генериране на EMS прозорци във всички адреси от адресируемата памет.
- Произволен размер на EMS прозорците.
- EMS страниците могат да бъдат защитени от случаен RESET.

Менажер на разширената памет (EMM)

Подобно на прекъсванията на ДОС 21H, което осигурява достъп до стандартния интерфейс на функциите на ОС, функциите на EMM могат да бъдат извикани чрез прекъсване 67H. Преди една програма да осъществи достъп до функция на EMM тя трябва да провери дали има инсталирана EMS. Ако програмата не направи това и се обърне към някоя функция то резултатът ще бъде напълно непредсказуем.Извикването може просто да не работи или системата да блокира. Проверката за наличието на EMS става като се провери зареждането на драйвера. Всеки драйвер има заглавен блок, а на отместване 10 от началото му е записано името на драйвера. LIM стандарта изисква това име да бъде EMMXXXXO. Ако има инсталиран EMM, сегментния адрес сочи към сегмента в, който е зареден драйвера. На отместване 10 от началото на сегмента се намира името на драйвера. Адреса на сегмента може да се получи, чрез функция XXH на ДОС.

След като това е проверено достъп може да се получи на три стъпки.

1. Отпускане на памет.Размерът на отпуснатата памет зависи от размера на разширената памет.
2. Ако желаният брой страници са успешно установени, определената страница трябва да бъде заредена в една от четирите страници на рамката за странициране, за да може да се осъществи четене и запис до страницата.
1. 3. Когато програмата завърши работа тя трябва да освободи заетата от нея памет. Както и при прекъсване 21H на ДОС номерът на функцията трябва да бъде записан в регистър AH преди извикването на прекъсването.

След извикването този регистър съдържа статуса на изпълнението на функцията.Резултат нула показва че няма грешка,а резултат по-голям от 80H показва номера на грешката.

Следните функции са необходими на програмите осъществяващи достъп до разширената памет:

40H Получаване на статуса на EMM.
41H Получаване сегментния адрес на рамката на страницата.
42H Получаване броя на страниците.
43H Заемане на EMS страници.
44H Задаване на пренасочването.
45H Освобождаване на страници от EMS.

За да сте сигурни, че EMS хардуера и EMM оперират по подходящ начин проверете статуса на на EMS преди да заявите разширена памет.

Ограничения в разпределението на разширената памет.

Броят на отпуснатите страници разширена памет е ограничен от броят на свободните страници. За да определи броя на свободните страници можем да използваме функцията 42H,която връща в регистъра BX броя свободни страници. Връща също и общия брой инсталирани страници в регистъра DX. При запазване на памет с функция 43H в регистъра BX се намира манипулатора на страницата.Този манипулатор трябва да се запази от програмата. Ако този манипулатор се загуби до страницата няма да има достъп и тя няма да може да се освободи. След като желаната логическа страница е в рамката на страниците достъпът до нея е като до нормална памет. Адресът на отместване от началото на страницата се определя от номера на физическата страница, която е в рамката на странициране, но съответният сегментен адрес трябва да бъде определен с функциите на EMM. Функцията 44H връща сегментния адрес на рамката за странициране.

Освен тези функции са налице и следните:

46H Получаване номера на версията.
47H Съхраняване на текущата карта.
48H Инициализиране на записаната карта.
49H Получаване на броя манипулатори ма EMM.
4AH Получаване броя на страници определени за манипулатор.
4BH Получаване на всички манипулатори и броя на заетите страници.

Функциите 47 и 48 са важни за резидентни програми, които искат да ползват разширената памет.Когато резидентната програма прекъсне някоя програма тя трябва да определи дали прекъснатата програма използва разширената памет и дали е създала някаква карта на паметта.Понеже пренасочването не трябва да бъде променяно от прекъсваната програма то трябва картата да бъде съхранена от резидентната програма и след това да се възстанови.

Руководство на потребителя

1. **Предназначение на библиотеката**

Библиотеката е предназначена за използване на Extended паметта в качеството на временен буфер за съхраняване на данни.
При недостиг на свободна Extended памет, или нейното отсъствие автоматично се изгражда своп файл на диска.
При това синтаксиса на командите не се изменя. Признак за създаване на своп файл е отрицателната стойност

на handle на заделения блок.

2. Изисквания към системата.

За работа на библиотека са необходими:

- IBM съвместимо PC;
- MS DOS 3.0 или по-висока
- при необходимост за използване на Extended във файла CONFIG.SYS е необходимо да се зареди HIMEM.SYS.

3. Описание на функциите

Функция	Подготовка на библиотеката за работа и инициализиране на променливите
Синтаксис	**extern int** EMS_Open(**void**)
Прототип	EMS_LIB.H
Описание	Проверява версията на DOS, включва адресната за да може да се използва Extended, запомня изходното състояние на системата,открива файла за свопинг.
Резултат	0 - успешно завършване 1 – системата не е инициализирана
Виж още	EMS_Close

Функция	Възстановява първоначалното съсстояние на системата
Синтаксис	extern **int** EMS_Close(**void**)
Прототип	EMS_LIB.H
Описание	Възстановява състоянието на системата до извикването на EMS_Open. Затваря свопинг файла.
Резултат	0 – при успех 1 - системата не е инициализирана
Виж също	EMS_Open

Функция	връща размера на най-големия свободен блок в килобайтове.
Синтаксис	**extern int** Get_Free_Size(**void**)
Прототип	EMS_LIB.H
Описание	Връща размера на най-големия свободен блок в Extended паметта. След извикването функцията променя Total_EXT показващ общия размер на достъпната Extended памет.
Резултат	При успех връща размера на най-големият свободен блок от Extended паметта. При неуспех EMS_Error съдържа код на грешка или 0.
Виж също	EMS_Alloc, EMS_Free, EMS_Realloc.

Функция	Заделя блок в Extended паметта.
Синтаксис	extern **long** EMS_Alloc(**int** Kbyte)
Прототип	EMS_LIB.H
Описание	заделя блок в Extended паметта,При грешка променя съдържанието на

EMS_Error. При извикване на функцията е необходимо да се укаже необходимия размер на паметта в килобайтове. Проверява свободната памет за блок с изисквания размер, ако няма такъв размер на блок се заделя памет в свопинг файла.

Резулта 0 – при грешка , иначе - handle на заделения блок от паметта

Виж също EMS_Free, EMS_Realloc.

Функция Променя размера на блок от Extended паметта.

Синтаксис extern **int** EMS_Realloc(unsigned **int** EMS_Handle, unsigned **int** Kbyte)

Прототип EMS_LIB.H

Описание Променя размера на заделен блок от Extended паметта. Функцията предава handle на блока и новия размер. Не се поддържа работа с файлове

Резултат 0 – при успех, иначе код на грешка.

Виж също EMS_Alloc, EMS_Free.

Функция Освобождава блок от Extended паметта.

Синтаксис extern **int** EMS_Free(unsigned **int** EMS_Handle)

Прототип EMS_LIB.H

Описание Освобождава по-рано зает блок от Extended паметта, или блок в свопинг файла. Функции предава handle на блока.

Резулта Връща 0 при успех, иначе връща код на грешка

Виж също EMS_Realloc, EMS_Alloc.

Функция Записва данни в Extended паметта.

Синтаксис extern **int** Send_To_Ext(unsigned **int** EMS_Handle,**char** *Ptr, unsigned **long** Byte)

Прототип EMS_LIB.H

Описание Прехвърля данни в Extended паметта или във файл. На функцията се предава handle на блока, в който се прехвърлят данните, адреса на данните и количеството на преместваните данни в байтове. Количеството на прехвърляните байтове трябва да бъде четно, в противен случай количеството на прехвърляните данни се увеличава с 1

Резултат Връща 0 при успех, иначе връща код на грешка

Виз също Send_To_Mem.

Функция Прехвърля данни от Extended паметта в конвенционалната памет.

Синтаксис extern **int** Send_To_Mem(unsigned **int** EMS_Handle, **char** *Ptr, unsigned **long** Byte)

Прототип EMS_LIB.H

Описание Прехвърля данни от Extended паметта или файл в конвенционалната памет. На функцията се предава handle на блока, от който ще се прехвърлят данните, адреса на данните в конвенционалната памет и количеството на преместваните данни в байтове. Количеството на прехвърляните байтове трябва да бъде четно, в противен случай количеството на прехвърляните данни се увеличава с 1

Резултат Връща 0 при успех, иначе връща код на грешка

Виж също Send_To_Ext.

Функция Получава номера на версията на XMS.
Синтаксис extern unsigned **int** Get_XMS_Ver(**void**)
Прототип EMS_LIB.H
Описание Получава номера на версията на XMS, и системна информация за наличие на HMA (1M to 1M + 64K): - променливата **Int**ernal_Rever съдържа верния вттрешен номер;- променливата HMA_Exist е равна на 1,ако съществува HMA, и 0, ако HMA не съществува. Не се поддържа работа с файлове
Резултат Връща номера на версията на XMS.

Функция заключва блок от Extended паметта.
Синтаксис extern **int** EMS_Lock(unsigned **int** EMS_Handle);
Прототип EMS_LIB.H
Описание Заключва блок от Extended паметта и връща неговия линеен адрес. Не се поддържа работа с файлове.
Резултат Връща 0 при успех, Addres_Line линеен адрес на локализирания блок инаше връща код на грешка.
Виж също: EMS_Unlock

Функция Отключва блок от Extended паметта.
Синтаксис extern **int** EMS_Unlock(unsigned **int** EMS_Handle);
Прототип в **EMS_LIB.H**
Описание Отключва блок от Extended паметта. Не се поддържа работа с файлове.
Резултат Връща 0 при успех, в противен случай код на грешка
Виж също EMS_Lock

Функция Получава информация за handle'ax.
Синтаксис extern **int** Get_Handle_Info(unsigned **int** EMS_Handle);
Прототип EMS_LIB.H
Описание Получава информация за handle'ax.: Block_Lock_Count –Брой на локализираните блокове; Num_Free_Hand_Left - количество свободни handle; Block_Size - размер на блока в килобайти. Не се поддържа работа с файлове.
Резултат Връща 0 при успех, в противен случай код на грешка.

4. Кодове за грешка

код	Грешка
01h	не коректна версия на DOS
02h	Свободната Extended памет е по-малко от изискваната
03h	Не може да се открие файл
04h	Грешка при запис във файл

80h неизпълнима функция
81h Открит е Vdisk
82h Грешка при включване на линии A20
8Eh Обща грешка в драйвера
8Fh Непоправима грешка в драйвера
90h HMA не е открит
91h HMA е вече зает
92h Стойността на DX е по малка от параметъра /HMAMIN (променете в CONFIG.SYS: HIMEM.SYS /HMAMIN=n)
93h невъзможно резервиране на HMA
94h невъзможно е да се включи линия A20
A0h Цялата Extended памет вече е резервирана
A1h Няма повече свободни манипулатори (handle) за блокове в Extended паметта (задайте в CONFIG.SYS повече стойности на n: HIMEM.SYS /HANDLES=ny
A2h Грешен handle
A3h Грешен handle на източник
A4h Грешен адрес на източника
A5h Грешен handle на блока на приемника
A6h Адреса на блока-приемник е грешен
A7h Грешен размер
A8h Прекъсване при грешка
A9h Грешно изпълнение на процеса
AAh Блока не е открит
ABh Блока е открит
ACh Препълване на брояча на локализираните блокове
ADh Локализирането невъзможно
B0h Има само малко в UMB
B1h Няма свободен UMB
B2h Грешен номер на сегмент в UMB

```
//---------------------------------------------------------------------------
// Декларации на функции упваляващи  Expanded Memory
 // Спецификация на  (EMS) функции.

/*--------------------------[ Прототипи на функции ]-------------------------*/

#if !defined(_MK_FP)
#define _MK_FP(seg,ofs)      ((void far *) (((unsigned long)(seg) << 16) | \
                    (unsigned)(ofs)))
#endif
int      _cdecl EmsAlloc(int numpages);
int      _cdecl EmsDealloc(int emmhandle);
int      _cdecl EmsExist(void);
unsigned _cdecl EmsFrame(void);
unsigned _cdecl EmsFree(void);
int      _cdecl EmsMap(int emmhandle,int logpage,int phypage);
int      _cdecl EmsRead(char *dest,unsigned emsofs,unsigned numbytes);
unsigned _cdecl EmsTotal(void);
char    *_cdecl EmsVer(void);
```

```
int      _cdecl EmsWrite(char *src,unsigned emsofs,unsigned numbytes);
unsigned _cdecl ExpMem(void);
unsigned _cdecl ExtMem(void);

//---------------------------------------------------------------------------
// Библиотека функции за работа с Expandet паметта
// Ems_Mem.c
//---------------------------------------------------------------------------
#include <dos.h>
#include "ems_mem.h"
static char *driver="EMMXXXX0";
//---------------------------------------------------------------------------
//  ЗАДЕЛЯ ПАМЕТ В  EMS memory
//---------------------------------------------------------------------------
int EmsAlloc(int numpages)
{
   union _REGS regs;
   regs.x.bx=numpages;
   regs.h.ah=0x43;           /* заделя  EMS страница */
   _int86(0x67,&regs,&regs);
   if(regs.h.ah) return(-1);  /* проверка за EMM грешка */
   return(regs.x.dx);         // връща EMM хендал(манипулатор) */
}
//---------------------------------------------------------------------------
// освобождава страница от  EMS паметта
//---------------------------------------------------------------------------
int EmsDealloc(int handle)
{
   union _REGS regs;
   regs.x.dx=handle;
   regs.h.ah=0x45;                     /* Освобождава EMS страница */
   _int86(0x67,&regs,&regs);
   return(regs.h.ah);                  /* връща  EMM код за грешка */
}

//---------------------------------------------------------------------------
// Определя дали е зареден  EMM драйвера
//---------------------------------------------------------------------------
int EmsExist(void)
{
   register int i,match=YES;
   char __far *p;
   union _REGS regs;
   struct _SREGS sregs;
   regs.x.ax=0x3567;              /* взема  EMS вектора на прекъсване */
   _int86x(0x21,&regs,&regs,&sregs);
   p=_MK_FP(sregs.es,10);          /* указател към EMM driver ID */
   for(i=0;i<8;i++) {             /* проверка за валидно име на драйвер */
      if(p[i]!=driver[i]) {
         match=NO;
```

```
            break;
         }
      }
   return(match);
}
//---------------------------------------------------------------------------
//  връща текущия  EMM фрейм сегмент
//---------------------------------------------------------------------------
unsigned EmsFrame(void)
{
   union _REGS regs;
   regs.h.ah=0x41;
    _int86(0x67,&regs,&regs);
   if(regs.h.ah) return(0);    /* проверка за  EMM грешка */
   return(regs.x.bx);          /* връща адреса на EMS страница  */
}
//---------------------------------------------------------------------------
//   Връща размера на свободните  EMS страници
//---------------------------------------------------------------------------
unsigned EmsFree(void)
{
   union _REGS regs;
   regs.h.ah=0x42;             /* Определя свободните страници */
   _int86(0x67,&regs,&regs);
   if(regs.h.ah) return(0);     /* проверява за  EMM грешка */
   return(regs.x.bx);
}

//---------------------------------------------------------------------------
// прехвърля логическа EMS страница във физическа DOS страница
//---------------------------------------------------------------------------
int EmsMap(int handle,int logpage,int phypage)
{
   union _REGS regs;
   regs.x.bx=logpage;
   regs.h.al=(unsigned char)phypage;
   regs.x.dx=handle;
   regs.h.ah=0x44;
   _int86(0x67,&regs,&regs);
   return(regs.h.ah);
}
//-------------------------------------------------------------
//      чете символ от  EMS паметта
//-------------------------------------------------------------
int EmsRead(char *dest,unsigned emsofs,unsigned numbytes)
{
   unsigned int emsseg,i;
   char __far *src;
   char *p;
   emsseg=EmsFrame();
```

```
   if(!emsseg) return(1);
   src=_MK_FP(emsseg,emsofs);
   p=dest;
   for(i=0;i<numbytes;i++) *p++=*src++;
   return(0);
}
//---------------------------------------------------------------------------
//   Връща общия брой EMS  страници
//---------------------------------------------------------------------------
unsigned EmsTotal(void)
{
   union _REGS regs;
   regs.h.ah=0x42;              /*  прочита свободните страници */
   _int86(0x67,&regs,&regs);
   if(regs.h.ah) return(0);      /*  проверка за  EMM грешка  */
   return(regs.x.dx);
}
//---------------------------------------------------------------------------
// връща версията на  EMS драйвера като текст
//---------------------------------------------------------------------------
char *EmsVer(void)
{
   static char *version="\0.\0\0";
   union _REGS regs;
   unsigned char ver;
   regs.h.ah=0x46;            /* определя EMS версията */
   _int86(0x67,&regs,&regs);
   if(regs.h.ah) return(NULL); /* проверка за EMM грешка */
   ver=regs.h.al;
   version[0]=(ver>>4)+0x30;  //премества битовете и ги преобразува в  ASCII
   version[2]=(ver&0x0f)+0x30;
   return(version);
}
//---------------------------------------------------------------------------
// Записва символ в EMS паметта   */
//---------------------------------------------------------------------------
int EmsWrite(char *src,unsigned emsofs,unsigned numbytes)
{
   unsigned int emsseg,i;
   char __far *dest;
   char *p;
   emsseg=EmsFrame();            /* ределя адреса на  EMS страницата */
   if(!emsseg) return(1);             /* проверка за  EMM грешка */
   p=src;
   dest=_MK_FP(emsseg,emsofs);
   for(i=0;i<numbytes;i++) *dest++=*p++;
   return(0);                        /* нормално завършване */
}
//---------------------------------------------------------------------------
//     връща размера  expanded паметта (ако я има)
```

```
//---------------------------------------------------------------------------
unsigned ExpMem(void)
{
   union _REGS regs;

   if(!EmsExist()) return(0);         /* проверява дали  EMM е инсталиран*/
   regs.h.ah=0x42;
  _int86(0x67,&regs,&regs);
   return(regs.x.dx*16);              /* връща определения размер */
}
//---------------------------------------------------------------------------
//    determines the amount of extended memory
//---------------------------------------------------------------------------
unsigned ExtMem(void)
{
   union _REGS regs;
   regs.x.ax=0x8800;                  /* определя размера на  extended паметта */
   regs.x.bx=0x8800;
   regs.x.cx=0x8800;
   regs.x.dx=0x8800;
   _int86(0x15,&regs,&regs);
   return((regs.x.dx<<8)+regs.x.ax);
}
```

4. Допълнителна памет.

За разлика от разширената памет която осигурява достъп на порции, поставянето на процесора в защитен режим осигурява достъп до цялата допълнителна памет. В реален режим допълнителната памет не може да се използва защото процесора може да адресира до 1M. Това обяснява защо системите базирани на процесорите 8086/8088 не могат да работят с допълнителна памет.Тези процесори работят само в реален режим.

Промяната само на един бит от флаговия регистър на процесора разрешава работата в защитен режим.Когато процесорът е в защитен режим той оперира с дескриптор на сегмента а не със сегментни адреси. Тези дескриптори сочат към локални или глобални списъци на сегментни дескриптори. Собствен дескриптор трябва да бъде създаден преди превключването на режима. Някой функции на BIOS и XMS драйвери предлагат възможност за достъп до допълнителната памет. Драйверите на XMS могат да управляват допълнителната памет по добре от BIOS. Тези драйвери осигуряват разпределение на допълнителна памет, вместо цялата допълнителна памет да обслужва само една програма. Най-ниските 64K от допълнителната памет са достъпни и в реален режим на адресация.

Достъп до допълнителната памет чрез BIOS

До допълнителната памет може да се достигне само ако тя съществува. Функцията 88H от прекъсване 15H на BIOS връща размера на допълнителната памет.Първоначално прекъсване 15H бе определено за достъп до касетофона. Когато дисковите устройства замениха касетофона неговите функции излязоха от употреба.Прекъсване 15H се използва за управление на допълнителната памет и джойстика. След като знаем че разполагаме с допълнителна памет ние можем да осъществим достъп до нея. Функция 87H премества

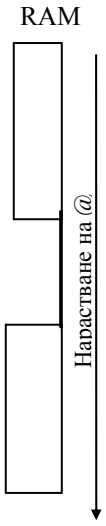
блок от паметта в цялото пространство на оперативната памет. Това означава, че може да се премества информация от конвенционалната памет в допълнителната памет и обратно.Функцията не бива да се използва за обратното защото използването и е сложно и има странични ефекти.

Глобална таблица на дескриптора

Двойката регистри ES:SI сочи адреса на глобалната таблица на дескрипторите (Global Descriptor Table или GDT), която трябва да бъде инсталирана от потребителската програма. GDT описва отделните сегменти от паметта на процесорите 80X86 в режим на защита. Сегментите в режим на защита се различават от тези в реален режим. Докато сегментите в реален режим могат да започват само от адрес кратен на 16, в защитен режим сегментите започват от произволно място. Освен това сегментите в защитен режим могат да имат различна големина от 1 байт до 64К. Друго нововъведение в режим на защита е кода за достъп, който е дефиниран за всеки сегмент. Този код показва дали описаният сегмент е за данни или за код на програмата. Кодът за достъп съдържа и информация за приоритетът на достъп. Всеки сегментен дескриптор се състои от 8 байта. По време на изпълнението си функция 87H очаква шестте сегментни дескриптора да са подготвени в таблица (GDT) т.е. има резервирана памет за тях. Фигура 9.2 илюстрира кои сегментни регистри се използват,а и конструкцията на сегментния дескриптор.

Адрес	Сегментен дескриптор
+0H	Дължина на сегмента
+2H	0-15 бита на сегмента
+4H	16-23 бита на сегмента
+5H	Код на достъп
+6H	Запазен
+8H	

GDT	0000 ADRESS
DVMMY	+00H
GDT	+08H
START	+10H
DEST.	+18H
BIOS CS	+20H
STACK	+28H
	+30H



фиг. 9.2

Когато се изпълнява тази функция (работа в защитен режим) всички прекъсвания трябва да се забранят. Докато процесорът е в защитен режим прекъсванията на BIOS (напр. таймер или клавиатура) могат да се използват, но те са проектирани само за работа в реален режим. Така, че прекъсванията могат да не работят по подходящ начин. При използването на защитен режим в следствие на забраната на прекъсванията системния часовник изостава спрямо реалното време. Необходимо е потребителската програма да осъществява корекции в системния часовник в съответствие с използваното процесорно време. Макар, че AT може лесно да се превключва в защитен режим обратния процес е доста сложен. За да се премине отново в реален режим е необходимо да се направи reset на процесора. В следствие на това

BIOS стартира програмата за начално зареждане. За да се избегне това първоначално зареждане оперативната памет използвана от BIOS получава код преди използването на reset. Този код информира BIOS за използването на последващия reset.След това BIOS връща управлението на програмата извикала функцията 87H. При системи оборудвани с процесор 80386 или по-високи този процес е много по лесен поради направените промени ни в процесора.

Конфликти в допълнителната памет.

На теория допълнителната памет трябва да се разпределя между програмите. Понякога обаче програмите например тези за кеширане на дисковите устройства изискват да използват цялата налична допълнителна памет. Това може да предизвика запис на информация върху други данни на други програми, което да доведе до колапс на системата. Проблемът произлиза от липсата на контрол.Необходима е програма която да помага на другите при разпределението на паметта Функция 88H информира, че цялата допълнителна памет е свободна. BIOS не може да запази отделни блокове памет. XMS стандарта предвижда няколко решения на този проблем.Този стандарт е разработен през 1988г. Две процедури помагат за избягване на конфликти при допълнителната памет, но не с постоянен успех. Първият и най-мощен метод (метода на прекъсване 15H) пренасочва вектора на прекъсване 15H така, че да сочи собствен манипулатор вместо оригиналния манипулатор на ROM-BIOS. Новият манипулатор трябва да се концентрира върху използването на функция 88H, която проверява размера на допълнителната памет. Манипулатора изпълнява винаги ROM-BIOS функцията която е заявена ако тя не е 88H.Ако е функция 88H резултатът не е цялата памет а само количеството свободна допълнителна памет.

Понеже извикващата програма допуска,че допълнителната памет започва от границата на един мегабайт,то инсталираните преди това програми трябва да се предпазят от нея, защото те допускат същото. Съществуващата програма се поставя в края на допълнителната памет вместо на границата на 1M. Извикващата програма възприема началото на програмата като край на допълнителната памет. Вторият метод на контрол на допълнителната памет е често използван от системните програмисти и има склонност към създаването на нови проблеми в програмирането. Този метод наречен VDISK бе използван първо при създаването на драйвера VDISK. VDISK е RAM диск, който може да използува допълнителната памет за съхраняване на файлове. VDISK съхранява данните си от границата на 1M вместо да се изолира от другите програми. VDISK не презаписва паметта резервирана чрез прекъсване 15H. Но всички програми заредени след него ще го презапишат вместо да използват свободната памет след него. Множеството RAM дискове усложняват достъпа до свободната допълнителна памет. Препоръчва се използването на XMS драйвера за достъп до допълнителната памет, защото такова изследване на свободната допълнителна памет не функционира на всички типове RAM дискове.

5. Директен достъп до HMA

Полето от високата памет (High Memory Area или HMA) са първите 64K от допълнителната памет, до която има достъп индиректно от реален режим без да се превключва в защитен. Как можем да достигне до паметта разположена извън адресното пространство на процеса?

Отговора може да бъде намерен в начина по който 80X86 определя физическия адрес в реален режим от адреса на сегмента и отместването в сегмента. Сегментния адрес се умножава по 16 и след това се събира с отместването. Ако изберем за сегмент последния

адрес от адресното пространство FFFFH, това изпраща адрес по голям от 000FH извън адресното пространство.Това ни дава възможност да адресираме допълнителната памет в реален режим. HMA започва от отместване 00010H вместо от 00000H затова тази памет е по-малка от 64K. Тя може да се използва за данни или за запис на резидентни програми.
Чрез командата DOS=UMB.

Стъпки необходими за достъп до HMA

Програмата, която иска да използва HMA областта трябва да се увери, че тя съществува. Ако процесорът е 80X86 можем да продължим, защото линията A20 е позволена и има допълнителната памет. След това чрез функция 15H се определя какъв е размерът на допълнителната памет. Достъпът до HMA е разрешен само ако допълнителната памет е минимум 64K. Адресната линия трябва да бъде предпазена от конфликти с контролера на клавиатурата и BIOS трябва да разреши адресна линия A20 преди да бъде осъществен достъп до HMA.Обикновено тази линия е забранена и това ви връща в началото на адресното пространство.

Адресно препълване със забранена адресна линия A20

Разрешаването на адресна линия A20 има върху контролера на клавиатурата същия ефект,както и превключването в режим на защита. При инициализацията линия A20 се превключва от изходния порт на контролера на клавиатурата. Бит 1 на този порт установява линия A20. Нормален персонален компютър с шина ISA (Indrustry Standart Architecture) използва изходния порт на клавиатурата за контролиране положението на адресна линия A20.При PS/2 системите и компютрите,оборудвани с процесор 80386,тази линия трябва да бъде освободена за други цели. Състоянието на линията трябва да бъде потвърдено с тест. В противен случай не може да се осъществи достъп до HMA паметта. За работа с HMA са необходими три процедури.

- Процедура,която определя наличието на най-малко 64к памет.
- Процедура,която превключва линия A20.
- Процедура,която проверява състоянието на линия A20 чрез сравняване на паметта (при блокирана линия A20 полетата с адрес FFFF:0010 до FFFF:FFFF съвпадат със съдържанието на паметта с адреси 0000:0000 до 0000:FFFF). За разлика от допълнителната памет HMA паметта не може да се раздава на части. Тя се получава изцяло от програмата, която я е заявила първа.Програмата получава или пълен достъп до паметта или този достъп е отказан.

Стандарт XMS

EMS стандарта съдържа стандартен софтуерен интерфейс за работа с разширената памет, а допълнителната памет не бе обхванатa от определен стандарт до 1988г., когато бе създадена Спецификация за Допълнителната Памет (eXtended Memory Spec**if**ication XMS). XMS стандартът дефинира софтуерен интерфейс, който позволява на множество програми да ползват допълнителната памет и други полета на паметта едновременно. Поддържат се следните типове достъп до паметта:

- HMA включваща първите 64K от паметта над 1M.
- Четири блока от допълнителната памет(Extended Memory Blocks), които са в допълнителната памет и започват от 1088K (Така се избягва конфликта с HMA).

- Горните блокове на паметта (Upper Memory Blocks или UMB), които са разположени в адресното пространство между 640K и 1 мегабайт. Най-широко разпространения драйвер на XMS стандарта е HIMEM.SYS на Microsoft Corporation. XMS интерфейса обикновено поддържа LIM стандарта.

Функциите на XMS се извикват от инструкция FAR CALL вместо с действително прекъсване.Следователно адресът на XMS манипулатора е необходим за извикването.Този XMS манипулатор се нарича Менажер на допълнителната памет (eXtendet Memory Manager или XMM).Прекъсване 2F връща този адрес само ако XMS драйвера е заредени и ако първото извикване на това прекъсване е върнало 80H в регистъра AL.В таблица 9.3 са показани осемнадесетте функции на XMS извиквани чрез прекъсване 2FH.

Функция	Предназначение
00H	Определяне номера на версията на XMS
01H	Заемане на полето от високата памет(HMA)
02H	Освобождаване на полето от високата памет(HMA)
03H	Глобално разрешение на адресна линия A20
04H	Глобална забрана на адресна линия A20
05H	Локално разрешение на адресна линия A20
06H	Локална забрана на адресна линия A20
07H	Въпрос за статуса на адресна линия A20
08H	Въпрос за свободната допълнителна памет
09H	Заемане на блок допълнителна памет(EMB)
0AH	Освобождаване на блок допълнителна памет(EMB)
0BH	Преместване на блок допълнителна памет(EMB)
0CH	Заключване на блок допълнителна памет(EMB)
0DH	Отключване на блок допълнителна памет(EMB)
0EH	Получаване на информация за манипулаторите на EMB
0FH	Промяна на размера на EMB
10H	Заемане на блок горна памет(UMB)
11H	Освобождаване на заявен блок горна памет(UMB)

таб. 9.3

При определяне на размера на допълнителната памет върнат от функция 08H е необходимо да се извади размерът на HMA блока от 64K за да се определи точният размер на свободната памет. Заявяването на блок допълнителна памет може да доведе до грешка независимо от това, че има свободна памет. Причината за грешката може да бъде манипулатора. Има фиксиран брой манипулатори и при изчерпването им е възможно да остане неизползвана памет.

При XMM се използва манипулатор а не директния адрес защото при работа с допълнителната памет нейните блокове се преместват динамично за да се се избегне фрагментацията.В следствие от това преместване малките блокове се групират в по големи при, което се освобождават компактни свободни полета от паметта.Този манипулатор трябва да бъде съхранен сигурно в противен случай при една случайна загуба заетата памет ще се освободи едва при рестартирането на системата. За разлика от конвенционалната памет допълнителната памет не се освобождава при излизане в DOS. Необходимо е тя да се освободи предварително с функция 0AH на 2FH прекъсване.

Понеже данните могат да бъдат записвани в UMB но не може да се манипулира с тях е необходимо да се прехвърлят често от допълнителната в конвенционалната памет и обратно. Това се извършва с функцията 09H.,

XMM предвижда и достъп до HMA .Както споменахме по горе този достъп се дава само на програмата заявила го първа. Поддържа се и високата памет UMB. Тази памет е разположена в рамките между 640K и 1M. Оперирането с тази памет е както при конвенционалната памет. За управлението и са включени две функции 10H за заделяне на блок и 11H за освобождаване на блок.

1. Управление на RAM от DOS.

Една от основните задачи на операционната система е управлението на RAM (Random Accsess Memory) паметта. Това е паметта в която всички системни програми,драйвери, TSR модули се използват заедно. Тъй като всички използват паметта то DOS трябва да осъществи съвместното и използване от програмите. Управлението на RAM от DOS се базира на заемането на блок от паметта с предварително определен размер. Този блок остава запазен за програмата до освобождаването му чрез дадени функции на DOS. DOS има четири различни функции за управление на паметта. 48H Заделяне на памет.

49H	Освобождаване на памет.
4AH	Промяна размера на блок от паметта.
58H	Четене/задаване режима на управление на паметта.

Функциите заделяне и освобождаване на паметта се използват от потребителските програми,както и от DOS при зареждане и изпълнение на приложения. Когато една програма се стартира DOS заделя част от паметта в която по късно програмата ще се зареди.Размерът на заделената памет зависи от вида на програмата,която трябва да се зареди. COM програмите резервират цялата RAM памет.Количеството памет която се заделя за EXE програми се взема от началната част на EXE файла. EXE файл може да се зареди само ако има достатъчно голям блок свободна памет. Освен от първоначално заетата памет доста приложения се нуждаят от допълнителна памет,която се заделя в процеса на работа на приложението.

#

2. Заделяне на памет.

Заделянето на памет се извършва с функцията 48H. Заделянето става в параграфи, като един параграф има размер от 16 байта. Следователно минималното количество памет, която може да се задели е 16 байта, а максималното количество 1M. Реално 1M памет не може да се задели, защото в наличност са само 640K при това част от нея е заета от системни области, драйвери и резидентни програми. Тъй като DOS не винаги може да задели необходимото количество памет е необходимо да се провери флага за пренос при всяко извикване на функция 48H. При вдигнат флаг DOS не е могъл да зареди необходимото количество памет.Върнатата стойност в регистъра BX дава количеството свободна памет. Тази способност на функцията ни дава възможност да определим количеството свободна памет. Ако поискаме да резервираме 1M памет, то DOS няма да може да я задели в следствие на това в регистъра BX ще се намира обема свободна памет. Преди

завършването на всяка програма допълнителната заета памет със функция 48H трябва да се освободи в противен случай DOS ще мисли, че тя е заета и няма да я заделя при последващо изискване за заделяне на памет. Освобождаването на блоковете памет става с функцията 49H. При правилно освобождаване на паметта функцията нулира флага за пренос в противен случай го вдига в единица и връща кода на грешката в регистъра AX.При някой програми е необходимо да се промени размера на заделената памет. Например при зареждането на резидентна програма е необходимо при завършването им с прекъсване 27H да се освободи излишната памет. Промяната на размера на блока памет става с функция 4AH.

Къде е разположена паметта?

Паметта заделена със функция 48H, произхожда от областта на временно ползване от потребителски програми. Това се отнася за паметта от края на резидентната част на DOS до края на конвенционалната памет (максимален размер 640K). Паметта между 640K и 1M както споменахме по горе се използва за ROM-BIOS, видео памет, входно изходни портове и потребителски нужди. Количеството UMB памет зависи от това колко място заема видео паметта.За разлика от конвенционалната памет тази памет е много фрагментирана. Понякога не може да се задели достатъчна по обем памет поради не разполагането и в съседни области т.е. UMB паметта е доста накъсана. За да се използва UMB от DOS програмите е необходимо те да се заредят в нея. Това става чрез командата разположена в COFIG.SYS:

DOS=HIGH,UMB или DOS=LOW,UMB

След намирането на една от тези команди DOS търси дали е зареден драйвера за разширената памет XMS).В действителност DOS не определя паметта в която да се зареди.Това се извършва от драйвера EMM386.EXE. Специални команди на DOS като DEVICEHIGH и LOADHIGH се използват за зареждане на програми и драйвери в горната част на паметта. UMB паметта може да се използва и от потребителски програми. Това става с функция 48H. Четири под-функции на функция 58H ви дават възможност да влияете на начина по който се управлява UMB паметта. Първите две функции съществуват от версия 2.0 , а вторите две от версия 5.0.

00H	Прочитане модела на паметта.
01H	Задаване модела на паметта.
02H	Прочитане състоянието на UMB.
03H	Задаване състоянието на UMB.

Ако искате да използвате UMB във вашата програма опитайте се да използвате малки блокове памет.

Модели на заделената памет.

Започвайки от версия 2.0, DOS позволява да се различен модел за заделяне на памет.Желания модел се избира с под-функция 01H на функция 58H. Този модел задава начина по който DOS търси блокове свободна памет. Различните възможности се определят по три начина:

00H	Търсене от долу на горе.
01H	Търсене на най-подходящата област.
02H	Търсене от горе на долу.

При функции 00H и 02H търсенето на свободни области от паметта започва от началото или от края на паметта. Ако първият намерен свободен блок не е по-малък от необходимия то той се заделя. При функция 01H се проверява цялата памет и се заделя блокът памет който е с размер най-близък до зададения но не е по малък от него. Тези кодове се записват в регистър BL при извикване на функция 01H.Освен това бит 7 има особено значение ако той е 1 търсенето ще започва от UMB, а ако е 0 от конвенционалната памет.

Същност на управлението на паметта.

Дотук говорихме за това как се заделя памет чрез функции на DOS и BIOS, сега ще покажем как е организирано физически управлението на паметта. Важна роля за управлението на паметта играе блока за управление на паметта (Memory Control Block или MCB). За управление на всеки зает блок от паметта с функция 48H се изгражда MCB блок, съдържащ 16 байта. MCB винаги започва от отместване кратно на 16 и предшества блока, който описва.Функциите за управление на паметта работят винаги с адреса на сегмента за заделения блок, така че сегмента на съответния MCB може да се намери, като извадим 1 от сегмента на заделения блок. MCB има следната структура:

+00H	ID ("Z"=последен MCB,"M"= следват още)	1 байт
+01H	Адрес на сегмента на съответстващото PSP	1 дума
+03H	Брой на параграфите в заделения блок	1 дума
+05H	не се използва	11 байта
+10H	Заделен блок от паметта	X параграфа

Първото поле показва дали блока е последен или има още MCB Второто поле съдържа адреса на PSP за съответната програма. Това поле има значение само ако заделения блок памет е за обкръжението на даден процес. В този случай то сочи адреса на префикса на програмата за която е заделено. Преди зареждането на програмата операционната среда заделя памет и изгражда обкръжението й. Ако блока представлява PSP то това поле сочи към блока. Третия блок служи за изчисляване на самия блок.Това поле дава размера на блока от паметта. Понеже веднага след този блок има друг MCB, то това поле дава положението на следващия MCB. По този начин всеки MCB сочи непряко следващия. За да се достигне до MCB е необходимо да се знае адреса на първия MCB.DOS съхранява този адрес във вътрешна структура наречена информационен блок (DIB DOS In**for**mation Block). Тази структура обикновено не е достъпна за потребителските програми.Обаче може да има достъп към нея с недокументираната функция 52H. Тази функция връща указател към информационния блок в двойката регистри ES:BX. Само че този адрес сочи към второто поле от таблицата а не към първото поле, което сочи адреса на първия MCB. Адреса на указателя към MCB се намира, като от отместването в сегмента получен от функция 52H се извади 4 т.е. адреса се сочи от ES:[BX-4]. Тази формула е валидна само ако отместването върнато от функция 52H е по-голямо от 3.

/* Извежда в прозорец на екрана съдържанието на първият заделен блок от паметта.
Забележка: За целта се използва функция 52h от 21-во прекъсване. Функцията връща адресът + 4 на първия MCB блок в регистрите ES:BX.*/

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
```

```
struct MCB {                    //  MCB
          unsigned char  id_code;    // 'M' = block follows, 'Z' = end
          unsigned int PSP;          // Сегментен адрес на приложението ( PSP)
          unsigned int Size;         // Брой резервирани параграфи
          };
void main( int argc, char *argv[], char *envp[])
 {
 unsigned char __far *p;
 struct MCB __far *cur_mcb;
 union _REGS in,ou;
 struct _SREGS seg;
 int  nr_mcb=1,j;
 in.h.ah=0x52;
 _int86x(0x21,&in,&ou,&seg);
 cur_mcb =(struct MCB __far *)MK_FP( seg.es-1, ou.x.bx+12 );
 cur_mcb =  ( struct MCB __far * )(*(long __far *)cur_mcb);
  printf("MCB number     = %d\n", nr_mcb);
  printf("MCB address    = %Fp\n", cur_mcb);
  printf("Memory address = %Np:0000\n", FP_SEG(cur_mcb)+1);
  printf("ID             = %c\n", cur_mcb->id_code);
  printf("PSP address    = %Fp\n", (unsigned char __far *) MK_FP(cur_mcb->PSP, 0) );
  printf("Size           = %u paragraphs ( %lu bytes )\n\n", cur_mcb->Size, (unsigned long) cur_mcb->Size << 4);
 p = MK_FP( FP_SEG(cur_mcb)+1,0);
 for( j=0;j < (cur_mcb->Size*16); printf("%3X",*p),p++,j++);
}
```

Microsoft C/C++ разполага със следните функции за динамична работа с паметта:

_alloca	_bmalloc	_fheapset	_heapmin	_nfree
_bcalloc	_bmsize	_fheapwalк	_nheapchк	_heapset
_bexpand	_brealloc	_heapwalк	_nheapmin	fmalloc
_bfree	calloc	_fmsize	_hfree	_nheapset
_bfreeseg	free	malloc	_nheapwalк	_expand
_bheapadd	_fcalloc	_memavl	_nmalloc	_frealloc
_bheapchк	_fexpand	_memmax	nmsize	_freect
_bheapmin	_ffree	halloc	_msize	_nrealloc
_bheapseg	_fheapchк	_heapadd	_ncalloc	realloc
_bheapset	_fheapmin	_heapchк	_nexpand	_stackavail
_bheapwalк				

Тези функции са описани в заглавния файл <malloc.h>. В тази част ще опишем синтаксиса на част от тях.

Име **malloc** – динамично заделяне на памет

Употреба **void** *malloc(size_t size);
size_t е тип дефиниран като **unsigned**
void *calloc(size_t nelem, size_t elsize);
За микро, малък и среден модел на паметта (tiny, small, medium):
unsigned coreleft(**void**); За компактен, голям и много голям модел на паметта (compact, large, huge):
void free(**void** *ptr);
void *realloc(**void** *ptr, **unsigned** newsize);

Прототип в stdlib.h и alloc.h

Описание Тези функции осигуряват достъп до динамично разпределяната памет на Си (областта heap). Там се отделя динамична памет за разполагане на променливи по обем блокове. Много структури от данни (напр. дървовидните, списъци и др.) естествено изискват такъв механизъм за отделяне на памет. При малките модели (микро, малък, и компактен) като динамична памет се използува мястото от края на сегмента за данни до ("върха" на програмния стек + 256 байта). Тези 256 байта се оставят като резерв за стека и за някои нужди на MS-DOS. При големите модели (среден, голям и много голям) цялото пространство над стека до физическия край на паметта е достъпно за динамично разпределение.

malloc връща указател към блок от паметта, с размер size. Ако няма достатъчно памет malloc връща NULL. Съдържанието на блока остава непроменено.

calloc отделя блок от памет както malloc, но размерът на блока е nelem елемента, всеки с размер elsize (действителният размер в байтове е nelem*elsize). Блокът се запълва с нули.

coreleft връща размера на свободната, неразпределена памет. Тя дава различни резултати, в зависимост от избрания модел на паметта.

free освобождава отделен преди това блок от паметта. ptr трябва да съдържа адреса на първия байт от блока.

realloc променя размера на блока на newsize, копирайки съдържанието на новото място, ако е необходимо.

Резултат **malloc** и **calloc** връщат указател към новоотделения блок или NULL, ако няма достатъчно място за новия блок.

realloc връща адреса на променения блок, който може да не съвпада с адреса на предишния. Ако не е възможна исканата промяна, realloc връща стойност нула. При големите модели на памет (среден, голям и много голям)

coreleft връща количеството неизползувана памет между края на динамично разпределената до момента (heap) и стека. При малките модели (микро, малък и компактен), coreleft връща количеството неизползувана памет между стека и сегмента за данни без 256 байта.

Пример:

```
#include  <stdio.h>
#include  <stdlib.h>
typedef struct {  /* ..... */   } OBJECT;
OBJECT *NewObject()
{
  return ((OBJECT *) malloc(sizeof(OBJECT)));
}
void FreeObject(OBJECT *obj)
{
        free(obj);
}
void main()
{
         OBJECT *obj;
        obj = NewObject();
        if (obj == NULL) {
                  printf("Не е създаден новият обект\n");
                  exit(1);
        }
         /* ..... */
        FreeObject(obj);
}
```

Име **_alloca** заделя памет
Синтаксис: **void** *_alloca(size_t size);
Виж също: _stackavail, malloc, calloc, new
Описание **_alloca** установява и отпуска <size> байта в програмния стек. Определя автоматично свободното мястото при извикването на функцията и излиза. Спазвайте следните ограничения когато използвате _alloca: ¨

- Когато компилирате с оптимизация ,указателят на стека може да не се възстанови и функцията трябва да възстанови подходящо функцията, която няма локални променливи и също така да даде справка за функцията _alloca . (Тези ограничения не се отнасят за DOS32X.)

Следващата програма илюстрира тези проблеми:

```
/* Компилиране с  CL /AM /Ox /Fc */
#include  <malloc.h>
void main( void )
{
   func( 10 );
}
void func( register int i )
{
   _alloca( i );
}
```

Резултат _alloca установява и връща **void** указател към заделената памет, която е подходящо подравнена с указания тип на обекта. Връща NULL ако не може да с задели памет.

Име **calloc** – динамично заделяне на памет
Прототип в: <malloc.h>, <stdlib.h>
Синтаксис: **void** *calloc(size_t num, size_t size);
void __based(**void**) *_bcalloc(__segment seg, size_t num, size_t size);
void __far *_fcalloc(size_t num, size_t size);
void __near *_ncalloc(size_t num, size_t size);
Резултат: Връщат указател към заделената памет при успех или NULL при неуспех. (_NULLOFF за _bcalloc при неуспех).
Функциите нулират заделената памет.
Виж също: malloc, free, _halloc, realloc, __based, new
Описание: **calloc** фамилията от функции заделят памет за масив от <num> елемента с размер <size> байта и го инициализират с 0.

За големите модели (compact-, large-, и huge-модел на програмата), calloc се заменя с _fcalloc.

За малки модели (tiny-, small-, и medium-програмен модел), calloc се заменя с _ncalloc.

Различните calloc функции заделят пространство от паметта показано в листинга по-долу. функция Heap сегмент calloc зависи от модела на данните на програмата

_bcalloc Based heap, специфициран от <seg> сегментен селектор
_fcalloc Far heap (извън подразбиращия се сегмент за данни

_ncalloc Near heap (вътре в подразбиращия се сегмент за данни)

Име **expand**

Синтаксис: **void** *_expand(**void** *memblock, size_t size);
void __based(**void**) *_bexpand(__segment seg,**void** __based(**void**) *memblock, size_t size);
void __far *_fexpand(**void** __far *memblock, size_t size);
void __near *_nexpand(**void** __near *memblock, size_t size);

Резултат: указател към новия блок при успех или NULL при неуспех. _bexpand връща _NULLOFF при грешка.

Виж също: realloc, malloc, free, new

Описание Функциите от фамилията _expand променят размера на предварително заделен блок от паметта като се опитва да разшири количеството, чрез преместване в локалния heap. <memblock> е указател към началото на блока. <size> показва новия размер на блока в байтове. Ако размера не може да се промени си остава стария размер. <memblock> може да бъде и указател към освободена преди това памет стига тя да е по-голяма по размер от зададената за да може да се извикат alloc, _expand, malloc, или realloc.ск> е сегментен адрес в базовия heap.

При голям модел на данните (compact-, large-, или huge), _expand се заменя с _fexpand. При малък модел на данните (tiny-, small-, или medium-), _expand се заменя с _nexpand. Различните _expand функции променят размера на блока с данни в сегмента за данни както е показано по-долу: Function Data Segment _expand Зависи от модела използван от програмата

_bexpand Based heap специфициран от <seg>, или във всички базови области ако <seg> е нула.

_fexpand Far heap (извън подразбиращия се сегмент за данни)

_nexpand Near heap (вътре в подразбиращия се сегмент за данни)

```
/* REALLOC.C илюстрира heap функциите:
 *  calloc  realloc    _expand
 */

#include  <stdio.h>
#include  <malloc.h>
#include  <stdlib.h>

void main()
{
  int  *bufint;
  char *bufchar;

  printf( "Заделя два буфера с размерност 512 елемента по два байта\n" );
  if( (bufint = (int *)calloc( 512, sizeof( int ) )) == NULL )
  exit( 1 );
  printf( "Отпуснати %d байта в  %Fp\n", _msize( bufint ), (void __far *)bufint );
  if( (bufchar = (char *)calloc( 512, sizeof( char ) )) == NULL )
  exit( 1 );
  printf( "Отпуснати %d байта в  %Fp\n",  _msize( bufchar ), (void __far *)bufchar );
  if( (bufchar = (char *)_expand( bufchar, 1024 )) == NULL )
     printf( "Can't expand" );
```

```
  else
  printf( "Expanded blocк to %d bytes at %Fp\n",    _msize( bufchar ), (void __far *)bufchar );
  if( (bufint = (int *)realloc( bufint,1024*sizeof(int))) == NULL)
     printf( "Can't reallocate" );
  else
    printf( "Reallocated blocк to %d bytes at %Fp\n",    _msize( bufint ), (void __far *)bufint );
  /* Освобождава паметта */
  free( bufint );
  free( bufchar );
  exit( 0 );
}
```

Име **_bheapseg**
Прототип в: <malloc.h>
Синтаксис: **int** _bfreeseg(__segment seg);
Резултат: 0 при успех или -1 при грешка.
Виж също: _bheapseg,__based,__segment,_bfree,_bmalloc,delete
Описание _bfreeseg освобождава базовия heap. Аргумента <seg> е върнатия базов heap при предишно извикване на _bheapseg.

```
/* HEAPBASE.C илюстрира динамично заделяне на памет в базовия heap
 *  _bheapseg   _bmalloc    _bfree  _bfreeseg */

#include  <stdio.h>
#include  <malloc.h>
#include  <stdlib.h>
#include  <string.h>

void main()
{
__segment seg;
  char __based( seg ) *outstr, __based( seg ) *instr;
  char __based( seg ) *pout,   __based( seg ) *pin;
  char tmpstr[80];  int  len;
  printf( "Enter a string: " );
  gets( tmpstr );
  /* Изисква базовия heap. */
  if( (seg = _bheapseg( 1000 )) == _NULLSEG )
  exit( 1 );
  /* Заделя в базовата  памет място за два стринга. */
  len = strlen( tmpstr );
  if( ((instr  = _bmalloc( seg, len + 1 )) == _NULLOFF) || ((outstr = _bmalloc( seg, len + 1 )) == _NULLOFF) )
     exit( 1 );
  /* Копира стринга с малки букви в динамичната памет. */
  _fstrlwr(_fstrcpy((char __far *)instr, (char __far *)tmpstr));
  /* Копира входния стринг в изходния в обратен ред. */
  for( pin = instr + len - 1, pout = outstr;  pout < outstr + len; pin__, pout++ )  *pout = *pin;  *pout = '\0';
  /* Показва стринга. */
  printf( "Input:  %Fs\n", (char __far *)instr );
  printf( "Output: %Fs\n", (char __far *)outstr );
  /* Освобождава басовия heap. */
_bfree( seg, instr );
_bfree( seg, outstr );
_bfreeseg( seg );
  exit( 0 );
}
```

Име **_heapadd**
Прототип в: <malloc.h>
Синтаксис : **int** _heapadd(**void** __far *memblocк, size_t size); **int** _bheapadd(__segment seg, **void** __based(**void**) *memblocк, size_t size);
Резултат : 0 при успех или -1 при грешка.
Виж също: _heapmin, _freect, _memavl
Описание _heapadd и _bheapadd добавят към употребяваната памет част от heap. _bheapadd добавя памет от специфицирания базов heap в <seg>. <size> специфицира размера който трябва да има вечег. _heapadd има стойността на сегмента ако той е в DGROUP,и добавя памет от near heap. В противен случай добавя памет от far heap.

Име **_heapchк**
Прототип в: <malloc.h>
Синтаксис: **int** _heapchк(**void**);
int _bheapchк(__segment seg);
int _fheapchк(**void**);
int _nheapchк(**void**);
Резултат: една от следните константи: _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEMPTY, _HEAPOK
Виж също: _heapset, _heapwalк
Описание _heapchк показва свързаните с heap проблеми при проверката за минимална устойчивост на heap.

Всяка функция проверява по части heap както следва:

Function Heap Checked

_heapchк зависи от модела на програмата
_bheapchк Based heap специфициран от <seg>
_fheapchк Far heap (извън подразбиращия се сегмент за данни) _nheapchк Near heap (в подразбиращия се сегмент за данни)

При голям модел (за compact-, large-, и huge-), _heapchк се заменя с _fheapchк. При малък модел (tiny-, small-, и medium), _heapchк се заменя от _nheapchк. Резултат:

Всичките четири връща обикновени цели числа е или следните константи които са дефинирани в MALLOC.H:
_HEAPOK _HEAPEMPTY _HEAPBADBEGIN _HEAPBADNODE

Име **_heapmin** освобождаване на паметта в heap.
Прототип в: <malloc.h>
Синтаксис: **int** _heapmin(**void**);
int _bheapmin(__segment seg);
int _fheapmin(**void**);
int _nheapmin(**void**);
Резултат: 0 при успех или -1 при грешка.
Виж също: _heapadd, _freect, _memavl

Описание Функциите от _heapmin фамилия минимизират heap като освобождават паметта в heap за операционната система.

Различните _heapmin функции освобождават памет както следва:

Функция	Минимизиране на хиипа
_heapmin	В зависимост от използвания модел
_bheapmin	Based heap специфициран в <seg>; _NULLSEG специфицира всички based heap
_fheapmin	Far heap (извън подразбиращия се сегмент за данни)
_nheapmin	Near heap (в подразбиращия се сегмент за данни)

При голям модел (за compact-, large-,и huge-), _heapmin се заменя с _fheapmin.
При малък модел (за small-, medium), _heapmin се заменя с _nheapmin.
За _heapmin, ако е избрана стойност за <seg> _NULLSEG, всички heap се минимизират, в противен случай само този който е показан.

Име **_heapset**
Прототип в: <malloc.h>
Синтаксис: **int** _heapset(**unsigned** fill);
int _bheapset(__segment seg, **unsigned** fill);
int _fheapset(**unsigned** fill);
int _nheapset(**unsigned** fill);
Резултат: една от следните константи: _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEMPTY, _HEAPOK
Виж също: _heapchk, _heapwalk
Описание Функциите от _heapset фамилията показ и отстраняват heap- свързаните проблеми,.

Име **_msize** показват свободната памет за заделяне
Прототип в: <malloc.h>
Синтаксис: size_t _msize(**void** *memblock);
size_t _bmsize(__segment seg, **void** __based(**void**) *memblock);
size_t _fmsize(**void** __far *memblock);
size_t _nmsize(**void** __near *memblock);
Резултат: размера в байтове.
Виж също: _memavl, _memmax
Описание Функциите от _msize фамилия връщат размера на блоковете от паметта заделени от функциите calloc, malloc, или realloc.
При голям модел (compact-, large-, и huge-модел на програмата), _msize се заменя с _fmsize.
При малък модел (tiny-, smalli medium- модел на програмата), _msize се заменя от _nmsize.

Име **realloc** промяна размера заделената памет
Прототип в: <malloc.h>, <stdlib.h>

Синтаксис **void** *realloc(**void** *memblocк, size_t size);
void __based(**void**)*_brealloc(__segment seg, **void** __based(**void**) *memblocк, size_t size);
void __far *_frealloc(**void** _far *memblocк, size_t size);
void __near *_nrealloc(**void** __near *memblocк, size_t size);

Резултат: указател към заделената памет при успех или NULL (_NULLOFF за _brealloc) при неуспех.

Виж също: _expand, malloc, calloc, free, new

Описание Функциите от фамилията realloc променят размера на преди товазаделен блок от паметта. <memblocк> аргумент е указател към началото на блока от паметта.
Ако <memblocк> е NULL (_NULLOFF за _brealloc), realloc функциите действа като malloc и заделя нов блок с размер <size> байта.

Име **_memmax** размера в байтове на максималната свобода памет

Прототип в: <malloc.h>

Синтаксис: size_t _memmax(**void**);

Резултат: размера в байтове на максималната свобода памет при успех или 0 при грешка.

Виж също: _msize, _memavl

Описание _memmax връща размера (в байтове) на най-големия съседен блок от паметта който може да се задели от near heap(текущият сегмент за данни). Например _nmalloc(_memmax) ще задели най-голямото количество памет.

Име **mem**... - действия с масиви от паметта (масиви, представящи блокове от определен брой поредни байтове от паметта)

Употреба **void** *memccpy(**void** *destin, **void** *source, **int** ch, size_t n);
void *memchr (**void** *s, **int** ch, size_t n);
int memcmp (**void** *s1, **void** *s2, size_t n);
int memicmp(**void** *s1, **void** *s2, size_t n);
void *memmove(**void** *destin, **void** *source, size_t n);
void *memcpy (**void** *destin, **void** *source, size_t n);
void *memset(**void** *s, **int** ch, size_t n);
size_t е тип дефиниран като **unsigned**.

Прототип в string.h mem.h

Описание Всички членове на фамилията работят с масиви, представящи блокове от паметта. Във всички функции n е размерът на масива в байтове.
memcpy копира блок от n на брой байта от мястото, определено от source в мястото, сочено от destin.
memmove работи аналогично на memcpy, но само тя обработва правилно застъпващи се блокове от паметта.
memset запълва всички байтове от областта, сочена от s, със символа ch. Дължината на запълваната област се определя от n.
memcmp сравнява n на брой байта от два символни низа, s1 и s2. Функцията сравнява байтовете като **unsigned char**, така че memcmp ("\xFF","\x7F",1) връща стойност по-голяма от нула.
memicmp сравнява първите n на брой байта от областите, сочени от s1 и s2, без да зачита разликата големи - малки букви от латинската азбука.

memccpy копира байтове от мястото, сочено от source, в мястото, сочено от destin. Копирането свършва при възникване на една от следните ситуации: ¨ Символът ch е копиран за първи път в destin ¨ Прекопирани са n на брой байта.

memchr търси символа ch в първите n на брой байта от масива s.

Резултат: **memmove** и **memcpy** връщат destin.

memset връща стойността на s.

memcmp и **memicmp** връщат стойност:< 0 ако s1 < s2 = 0 ако s1 = s2 > 0 ако s1 > s2

memchr търси символа ch в първите n на брой байта от масива s.

Ако ch е копиран, memccpy връща указател към байта от destin, следващ символа ch. В противен случай memccpy връща NULL.

memchr връща указател към първото появяване на ch в s. Ако ch не е намерен в масива s, резултатът е нула

.

Име **movedata** - копира порция байтове

Употреба **void** movedata(**unsigned** segsrc, **unsigned** offsrc, **unsigned** segdest, **unsigned** offdest,size_t numbytes);

Прототип в mem.h string.h

Описание movedata копира numbytes на брой байта от източник с адрес segsrc:offsrc, в мястото, определено от адрес segdest:offdest. В movedata е полезна при преместване на данни, разположени в други сегменти, ако използуваният модел на паметта е микро, малък или среден, където адресите на сегмента за данни не са известни явно. За моделите компактен, голям и много голям може да се използува memcpy, тъй като там адресът на сегмента е известен явно.

Резултат Функцията не връща резултат.

Пример:

```
#include  <mem.h>
#define MONO_BASE 0xB000
/* Запазва съдържанието на монохромния екран в буфер */
void save_mono_screen(char near *buffer)
{
  movedata(MONO_BASE,0,_FP_SEG(buffer),_FP_OFF(buffer),80*25*2);
}
void main()
{
  char buf[80*25*2];
  save_mono_screen(buf);
 }
```

Име **movemem** - премества блок от байтове

Употреба **void** movmem(**void** *source,**void** *destin,**unsigned** len);
void setmem(**void** *addr,**unsigned** len,**char** value);

Прототип в mem.h

Описание **movemem** копира блок от len на брой байта от source в destin.
Ако двете области се застъпват, посоката на копиране се избира така, че копирането да е коректно. **S**
etmem запълва първите len на брой байта от блока, сочен от addr със стойността value.

Резултат movemem и setmem не връщат резултат.

Начини за предаване на данни между програми на Си.

Малко са програмите , които не предават и/или получават информация от други процеси или от DOS ниво. Когато е необходимо, за това могат да се използват следните методи:

1. Предаване на параметри от командния ред.
2. Използване на кода на завършване/завръщане.
3. Предаване на параметри чрез обкръжението.
4. Използване на стека.
5. Използване на комуникационната област на DOS
6. Използване на общодостъпна област от RAM или файл.
7. Предаване на параметри, чрез въвеждане на собствени функции на прекъсване от DOS или BIOS.
8. Използване на дисков буфер за предаване на данни.

Описание на методите.

1. Предаване на данни от командния ред.

Под параметри на командния ред се разбира информацията между края на името на програмата и <ENTER>. Това е основният метод за предаване на данни от ниво на DOS. Параметрите трябва да са разделени от името на програмата с една или няколко празни позиции, или символи които не могат да участват в името на файла. Чрез параметрите от командния ред може лесно да се управлява логиката на потребителската програма. Ако параметрите са данни те се обработват с максимална бързина,т.е не е необходимо потребителя да ги въвежда в процеса на работа. За достигане до данните от командния ред в Си програми се използват параметрите на главната функция main(). Тя има следния синтаксис:

```
main( int argc, char *argv[], char *envp[] )
  {
        program-statements
  }
```

argc - Променлива от тип **int** съдържаща броя на параметрите от командния ред. Ако няма зададени параметри стойността и е 1.

argv - Масив от указатели сочещи към параметрите на командния ред.Първият указател дава пълната файлова спецификация на процеса.

envp - Масив от указатели сочещи към променливите на обкръжението. За край на обкръжението се разбира празен низ.

Пример:

```
#include <stdio.h>
#include <process.h>
void main( int argc,                /* Брой елементи в командния ред */
           char *argv[],    /* Аргументи на командния ред    */
           char **envp )    /* Променливи от обкръжението    */
{
   int count;
   /* Извежда параметрите на командния ред*/
   printf( "\nCommand-line arguments:\n" );
   for( count = 0; count < argc; count++ )
     printf( "  argv[%d]   %s\n", count, argv[count] );
   /* Извежда съдържанието на обкръжението */
   printf( "\nEnvironment variables:\n" );
   while( *envp != NULL )
     printf( "  %s\n", *(envp++) );
   printf( "\nProcess id of parent: %d", _getpid() );
   exit( 0 );
}

/*
 Да се направи програма на С, която да изпълнява породен процес.
Породения процес да  изчисли сумата на  две целочислени числа.
Числата да се въведат в родителския процес и да се предават като параметри
на командния ред. Породения процес да върне на родителския процес сумата.
*/
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
void main( int argc, char *argv[], char *envp[])
{
 int i,j,k;
 char Str[6],Str1[7];
 printf("\nInput i: "); scanf("%d",&i);
 printf("\nInput j: "); scanf("%d",&j);
  _itoa(i,Str,10);
  _itoa(j,Str1,10);
 k=_spawnl(_P_WAIT,"zad40b","",Str,Str1,NULL);
  printf("\nSum: %d",k);
  getchar();
}

#include <stdio.h>
#include <stdlib.h>
int main( int argc, char *argv[], char *envp[])
{
  if ( argc < 3 )
  {
    printf("\n Syntax : %s  i  j",argv[0]);
    return 0;
  }
   return ( atoi(argv[1]) + atoi(argv[2]));
}
```

2. Използване на кода на завършване.

След завършването на една програма DOS записва 2 байта информация в своята работна област за това:

- 1 байт указващ как е завършила програмата, т.н. байт за типа на завършване " termination type"
- 1 байт за кода на завръщане в DOS- "**return** code".

Двата байта могат да се прочетат еднократно чрез FUN4DH на DOS.

FUN 4Dh (DOS 2+) , Get **Return** Code
вход : AH= 4Dh
изход: AH тип на завършване:
0-нормално
1- програмата е била прекъсната чрез Ctrl/C
2- прекъсната поради критична грешка **int** 24h
3- завършила е нормално като е останала резидентна AL код на завръщане

Кодът на завръщане е число от 0_FFh, което се определя от програмиста при приключване на програма чрез fun 4Ch или fun 31h. Прието е, че код на завръщане 00 означава нормално завръщане.

В Си програми завръщането се извършва чрез функциите

void exit(**int** status);
void _exit(**int** status);
void _cexit(**void**);
void _c_exit(**void**);

По надолу са показани и функциите за изпълнение на породени процеси в Си програми които връщат и статуса на завръщане.

3. Предаване на параметри чрез обкръжението.

При стартиране на процес в средата на MS-DOS за него се отде- лят два блока от паметта:
- за изпълнимия код на програмата
- за обкръжението й.

Обкръжението е област от паметта която е с размер от 160 байта до 32768 байта и има следния формат:

"име на променлива1 = стойност" ,0
"име на променлива2 = стойност" ,0
.....
"име на променлива N = стойност" ,0
нулев байт.

Достъп до обкръжението освен, чрез параметрите на главната функция може да се осъществи и чрез функциите:

```
char *getenv( char *varname );
int  _putenv( char *envstring );
void _searchenv( char *filename, char *varname,char *pathname );


 /* ENVIRON.C илюстрира функциите за работа с обкръжението:
 *  getenv  putenv  _searchenv
 */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

void main( void )
```

```
{
   char *pathvar, pathbuf[256], filebuf[256];

   /* Взема  PATH от обкръжението и го съхранява. */
   pathvar = getenv( "PATH" );
   strcpy( pathbuf, pathvar );
   printf( "Old PATH: %s\n", pathvar ? pathvar : "variable not set");

   /* Добавя новата директория в пътя */
   strcpy( pathbuf, "PATH=" );
   strcat( pathbuf, pathvar );
   strcat( pathbuf, ";\\C700\\INIT;" );
   if( _putenv( pathbuf ) == -1 )
   {
    printf( "Failed\n");
    exit( 1 );
   }
   else
    printf( "New PATH: %s\n", pathbuf );

   /* Търси файла в новия път */
   _searchenv( "TOOLS.INI", "PATH", filebuf );
   if( *filebuf )
    printf( "TOOLS.INI found at %s\n", filebuf );
   else
    printf( "TOOLS.INI not found\n" );

   /* Възтановява оригиналния път */
   strcpy( pathbuf, "PATH=" );
   strcat( pathbuf, pathvar );
   if( _putenv( pathbuf ) == -1 )
    printf( "Failed\n");
   else
    printf( "Old PATH: %s\n", pathvar );
   exit( 0 );
}
```

Освен чрез променливите от обкръжението данни могат да се предадат и по следният начин. В първите два байта след края на променливите от обкръжението се записва размера на данните в байтове. След което следват данните.

/* Да се направи програма на C, която да изпълнява породен процес.
Породения процес да изчисли сумата на две целочислени числа.
Числата да се въведат в родителския процес и да се предават като променливи от обкръжението. Породения процес да върне на родителския процес сумата.*/

```
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
void main( int argc, char *argv[], char *envp[])
{
 char Str[7],Str1[7];
 char *Env[] ={Str,Str1,NULL};
 int i,j,k;
 printf("\nInput i: "); scanf("%d",&i);
 printf("\nInput j: "); scanf("%d",&j);
 sprintf(Str,"%6d",i);
 sprintf(Str1,"%6d",j);
 k=_spawnle(_P_WAIT,"zad41b",NULL,Env);
 printf("\nSum: %d",k);
 getchar();
```

```
}

#include <stdio.h>
#include <stdlib.h>
int main( int argc, char *argv[], char *envp[])
{
  return ( atoi(envp[0]) + atoi(envp[1]));
}
```

4 Предаване на параметри чрез стека.

5. Използване на комуникационната област на DOS

Тази област е в сегмента за данни на DOS- сегмент 0040H. 16 байта от отместване F0h от началото на сегмента са резервирани за потребителски цели.Тази област може да се използва от програма, която трябва да предаде данни на друга. Достъпът до тази област е неограничен.

6 Използване на общодостъпна област от данни или файл.

За да се използва общодостъпен буфер може да се процедира по следния начин:

Създава се резидентна програма която се дефинира като нова функция на DOS или BIOS. Тази функция създава резервира блок от паметта при първоначалното си зареждане и при всяко следващо извикване връща адреса на буфера от паметта. Потребителските програми за да комуникират помежду си извикват тази функция и чрез върнатата от нея стойност се разбира адресът на буфера. Друг начин за комуникиране е като се използва общ файл за данни.

```
/*
 Да се направи програма на C, която да изпълнява породен процес.
Породения процес да изчисли сумата на две целочислени числа.
Числата да се въведат в родителския процес и да се предават
посредсвом вектор на прекъсване 255. Породения процес да върне на
родителския процес сумата.
*/
#include <process.h>
#include <stdio.h>
void main( int argc, char *argv[], char *envp[])
{
    int __far *p =(int __far *)0x3fc;
    int k;
    printf("\nInput i: "); scanf("%d",p);
    printf("\nInput j: "); scanf("%d",(p+1));
     k=_spawnl(_P_WAIT,"zad42b",NULL);
    printf("\nSum: %d",k);
    getchar();
}


#include <stdio.h>
#include <dos.h>
int main( int argc, char *argv[], char *envp[])
{
  int __far *p =(int  __far *)0x3fc;
  return (*p + *(p+1));
}
```

Функции изпълняващи породен процес.

Тези функции изпълняват породен процес. Необходимите параметри са: cmdname име на файла който ще се изпълнява. По подразбиране функцията търси файлове с разширение COM,ако не го открие търси файл с разширение EXE.

Допълнителните букви в името на функцията означават **l, v, p** и **e** характеризират функция с определени възможности:

буква	значение
p	Функцията ще търси породения процес в директориите,посочени в променливата PATH от обкръжението на ДОС. Функциите без наставка p търсят само в главната и в текущо активната директории.
l	Показва, че аргументите указатели се предават на функцията като отделни аргументи arg0, arg1,...,argn. Обикновено функциите с наставка l се използуват, когато броят на предаваните аргументи е известен предварително
v	Показва, че аргументите arg[0],..., arg[n] се предават като масив от указатели. Тези функции се използуват при променлив брой аргументи.
e	Показва, че на породения процес може да се предаде аргумента envp и следователно да е възможна промяна на обкръжението за породения процес. При функциите без тази наставка, породеният процес наследява обкръжението от основния (родителския).

Име **execl** Тези функции изпълняват породен процес

Синтаксис:
```
int _execl(char *cmdname,char *arg0,... char *argn,NULL );I
nt _execle(char *cmdname, char *arg0, ... char *argn, NULL, char **envp );
int _execlp( char *cmdname,char *arg0,... char *argn,NULL)
int _execlpe( char *cmdname, char *arg0, ... char *argn, NULL, char **envp );
int _execv( char *cmdname, char **argv );
 int _execve( char *cmdname, char **argv, char **envp );
int _execvp( char *cmdname, char **argv );
int _execvpe( char *cmdname, char **argv, char **envp );
```

Резултат: -1 при грешка. errno: E2BIG, EACCES, EMFILE, ENOENT, ENOEXEC, ENOMEM

E2BIG	Твърде дълъг списък от аргументи
EINVAL	Невалиден аргумент
ENOENT	Ненамерен път или файл
ENOEXEC	Грешка във формата за изпълнение
ENOMEM	Няма достатъчно памет

Виж също: _execv..., _spawn..., system

Прототип <process.h>, <errno.h>

Описание : Тези функции изпълняват породен процес

Кодовете за грешки имат следното значение:	
Константа	Значение
ECHILD	Липсва породен процес

EAGAIN	Няма повече процеси.
E2BIG	Листът от аргументи е много дълъг
EACCES	Permission denied.
EBADF	Грешен номер на файл
EDEADLOCK	Resource deadlock would occur.

Име **system** - изпълнява команда от MS-DOS
Прототип <process.h>, <stdlib.h>, <errno.h>.
Синтаксис: **int** system(**char** *command);
Описание **system** извиква системния команден процесор на MS-DOS command.com за да изпълни комндата, определена чрез аргумента command. Действието се извършва все едно, че командата е въведена в отговор на знака за готовност на MS-DOS. За намиране на command.com се използува променливата
COMSPEC от обкръжението на операционната система.
Резултат. След изпълнението на командата функцията връща състоянието при излизане за COMMAND.COM (exit status). Не нулево число ако <command> е NULL и командният интерпретатор е в наличност. Ако командният интерпретатор липсва връща 0 и грешка ENOENT. Ако <command> не е NULL, връща 0 ако командният интерпретатор е стартиран успешно. -1 индицира за грешка. errno: E2BIG, ENOENT, ENOEXEC, ENOMEM
Виж също: _execl..., _spawn

Име **spawnl** Създават и изпълняват нов породен процес.
Прототип <process.h>, <stdio.h>, <errno.h>
Синтаксис: **int** _spawnl(**int** mode, **char** *cmdname, **char** *arg0, **char** *arg1,...**char** *argn, NULL);
int _spawnle(**int** mode, **char** *cmdname, **char** *arg0, **char** *arg1,...**char** *argn, NULL, **char** **envp);
int _spawnlp(**int** mode, **char** *cmdname, **char** *arg0, **char** *arg1,...**char** *argn, NULL);
int _spawnlpe(**int** mode, **char** *cmdname, **char** *arg0, **char** *arg1,...**char** *argn, NULL, **char** **envp);
int _spawnv(**int** mode, **char** *cmdname, **char** **argv);
int _spawnve(**int** mode, **char** *cmdname, **char** **argv, **char** **envp);
int _spawnvp(**int** mode, **char** *cmdname, **char** **argv);
int _spawnvpe(**int** mode, **char** *cmdname, **char** **argv, **char** **envp);
mode: _P_OVERLAY, _P_WAIT
Резултат: породения процес връща статуса си (mode = _P_WAIT) при успех или -1 при неуспех. errno: E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM
Описание Функциите от фамилията spawn... създават и изпълняват породени процеси. За да се зареди и изпълни файл като породен процес, системата трябва да разполага с достатъчно памет. Стойността на mode определя действието на основния процес (родителския) след обръщението към spawn. Дефинираните стойности са:
P_WAIT Основният (родителски) процес спира до завършване изпълнението на породения.
P_NOWAIT Основният процес продължава да работи заедно с породения.

P_OVERLAY Породеният процес се зарежда на мястото на основния (родителския), също както обръщение към exec..
ЗАБЕЛЕЖКА: В настоящата реализация стойността P_NOWAIT не е достъпна. Тя е предвидена за бъдещи разширения на системата и използването й сега ще генерира грешка.
pathname е името на файл, съдържащ програмата, която ще работи като породен процес.

Функцията spawn.търси файла, като използува стандартния алгоритъм на ДОС за търсене:

- Ако няма разширение или точка, търси зададеното име. При неуспех се добавя ".exe" и търсенето се подновява.
- Ако е зададено разширение, ще се търси файл със зададеното име.
- Ако името завършва с точка, ще се търси файл с това име и без разширение.

Всяка функция spawn... трябва да има една от наставките, определящи вида на предаване на аргументите (l или v). Наставките за начина на търсене на файла (p) и за наследяване на обкръжението (e) са незадължителни.

При функциите spawn... на породения процес трябва да се предаде поне един аргумент (arg0 или argv[0]). Този аргумент по подразбиране съдържа копие на pathname (използуване на друга стойност би предизвикало грешка).

При работа с MS-DOS 3.0 и следващи версии, pathname е достъпен за породения процес. При предишните версии на ДОС, породеният процес не може да използува този аргумент.

Когато функцията има наставка l, arg0 обикновено сочи pathname, а arg1,...,argn сочат символни низове, които формират новия списък от аргументи. Задължителният аргумент NULL след argn отбелязва края на списъка.

При наставка e чрез аргумента envp потребителят предава списък от нови стойности за обкръжението. Аргументът е масив от **char***. Всеки негов елемент сочи нулево прекъснат символен низ с формат:

ПромОтОбкръжението = Стойност

така се задава името на променлива от обкръжението и нейната стойност. Последният елемент в envp[] е NULL. Когато envp[0] е NULL, породеният процес наследява обкръжението от родителския.

Общата дължина на arg0+arg1+...+argn (или argv[0]+argv[1]+...+argv[n]), включително и интервалите, разделящи аргументите трябва да бъде по-малка от 128 байта. Не се броят символите '\0'.

При обръщение към функция spawn... всички отворени файлове остават отворени за породения процес.

Резултат При успешна работа функцията връща статуса на завършване на породения процес (0 при нормален край).

При грешка функциите spawn... връщат -1 и errno получава някоя от следните стойности:

E2BIG	Твърде дълъг списък от аргументи
EINVAL	Невалиден аргумент
ENOENT	Ненамерен път или файл

ENOEXEC	Грешка във формата за изпълнение
ENOMEM	Няма достатъчно памет

```
/* SPAWN.C illustrates the different versions of spawn including:
 * _spawnl    _spawnle          _spawnlp          _spawnlpe
 * _spawnv              _spawnve          _spawnvp          _spawnvpe
 *
 * Although SPAWN.C can spawn any program, you can verify how different
 * versions handle arguments and environment by compiling and
 * specifying the sample program ARGS.C. See EXEC.C for examples
 * of the similar exec functions.
 */

#include  <stdio.h>
#include  <conio.h>
#include  <process.h>

char *my_env[] =      /* Environment for _spawn?e */
{
   "THIS=environment will be",
   "PASSED=to child by the",
   "SPAWN=functions",
   NULL
};

void main()
{
   char *args[4], prog[80];
   int ch, r;
   printf( "Enter name of program to spawn: " );
   gets( prog );
   printf( " 1. _spawnl   2. _spawnle   3. _spawnlp   4. _spawnlpe\n" );
   printf( " 5. _spawnv   6. _spawnve   7. _spawnvp   8. _spawnvpe\n" );
   printf( "Type a number from 1 to 8 (or 0 to quit): " );
   ch = _getche();
   if( (ch < '1') || (ch > '8') )      exit( -1 );
   printf( "\n\n" );
   /* Arguments for _spawnv? */
   args[0] = prog;
   args[1] = "_spawn??";
   args[2] = "two";
   args[3] = NULL;

   switch( ch )
   {
    case '1':  r = _spawnl( _P_WAIT, prog, prog, "_spawnl", "two", NULL );break;
    case '2':  r = _spawnle( _P_WAIT, prog, prog, "_spawnle", "two",NULL, my_env );break;
    case '3':  r = _spawnlp( _P_WAIT, prog, prog, "_spawnlp", "two", NULL );break;
    case '4':  r = _spawnlpe( _P_WAIT, prog, prog, "_spawnlpe", "two", NULL, my_env );  break;
    case '5':  r = _spawnv( _P_WAIT, prog, args );  break;
    case '6':  r = _spawnve( _P_WAIT, prog, args, my_env );  break;
    case '7':  r = _spawnvp( _P_WAIT, prog, args );  break;
    case '8':  r = _spawnvpe( _P_WAIT, prog, args, my_env );  break;
   default:  break;
  }
   if( r == -1 )  printf( "Couldn't spawn process" );
   else  printf( "\nReturned from SPAWN!" );
   exit( r );
}
```

Име exit - прекъсва програмата

Употреба **void** exit(**int** status);
void _exit(**int** status);

Прототип в process.h

Описание **exit** прекъсва процеса, който се обръща към нея. Преди завършване всички файлове се затварят, буфериранияят изход се записва и ако чрез atexit са дефинирани функции, които оформят завършването на програмата, те ще бъдат извикани.
_exit прекъсва процеса без да затваря файловете, без да записва буферираните данни и да извиква функции, дефинирани от atexit.
Във всички случаи извикващият процес получава данни за прекъсването чрез status.
Стойност 0 показва нормално прекъсване на процеса, а стойност различна от 0 показва грешка.

Резултат Функциите не връщат стойност.

```
//    Изпълнява породен процес. Породеният процес  извежда в прозорец на екрана  съдържанието
// на PSP на родителския процес. Адреса на PSP да се предава, като  параметър на командния ред.
*
#include <process.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
void main()
{
    char Str[12];
   _itoa(_psp,Str,10);
   printf("%x %s ",_psp,Str)   ;
    _spawnl(_P_WAIT,"zad52b","",Str,NULL);
}

#include <stdio.h>
#include <string.h>
#include <graph.h>
#include <stdlib.h>
#include <dos.h>
int main( int argc, char *argv[], char *envp[])
{
 unsigned  char __far *p;
  char Str[225];
  int i;
  // unsigned int  PSP_SEG;
  if ( argc < 2 )
  {
   printf("\n Syntax : %s  PSP_ADDRES",argv[0]);
   return 1;
  }
     _settextwindow(5,10,20,70);
      _setbkcolor(1);
      _clearscreen(_GWINDOW);
     p = (unsigned  char __far*)_MK_FP( atoi(argv[1]),0);
    sprintf (Str,"-- %s  ---- %x       ",argv[1],_FP_SEG(p));
   _outtext(Str);
    _settextposition(2,1);
    for( i = 0; i < 256; i++)
```

```
    {
      sprintf(Str,"%3X",*(p+i));
     _outtext(Str);
     }
 return 0;
}

//    Изпълнява породен процес. Породеният процес  извежда в прозорец на екрана  съдържанието
// на PSP на родителския процес. Адреса на PSP да се предава, като  елемент от обкръжението.

#include <process.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
void main()
{
    char Str[40]="PSP=";
    char *Env[]= {NULL,NULL};
    _itoa(_psp,(Str+4),10);
    Env[0] = Str;
   printf("%d |%s ",_psp,Str[0])   ;
    _spawnle(_P_WAIT,"zad53b",NULL,Env);
}

#include <stdio.h>
#include <string.h>
#include <graph.h>
#include <stdlib.h>
#include <dos.h>
int main( int argc, char *argv[], char *envp[])
{
 unsigned  char __far *p;
  char Str[225];
  int i=0;
   while( envp[i][0] != '\0')
  {
   if ( strncmp( envp[i] ,"PSP",3) == 0)
   {
      // _setvideomode(_TEXTC80);
      _settextwindow(5,10,20,70);
      _setbkcolor(1);
      _clearscreen(_GWINDOW);
      _FP_SEG(p) = atoi( (envp[i]+4));
    _FP_OFF(p)  = 0;
    sprintf (Str,"-- %s  ---- %d       ",(envp[i]+4),_FP_SEG(p));
   _outtext(Str);
    _settextposition(2,1);
    for( i = 0; i < 256; i++)
    {
      sprintf(Str,"%3X",*(p+i));
     _outtext(Str);
     }

         getchar();
         return 0;
    }
    i++;
    }
    printf("\n Syntax : %s  PSP_ADDRES",argv[0]);
   return 1;
```

```
}

//    Изпълнява породен процес. Породеният процес  извежда в прозорец на екрана  съдържанието
// на PSP на родителския процес.  Адреса на PSP да се предаде посредством вектор на прекъсване 255.
*
#include <process.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
void ( _cdecl __interrupt __far *Int_255)();
void main()
{
     long __far *p =(long __far *)0x3fc;
    _FP_SEG(Int_255) = _psp;
    _FP_OFF(Int_255) = 0;
    _dos_setvect( 0xFF, (void ( __interrupt __far *)() )(*Int_255));
     printf("%Fp     | %Fp  -- %x -- %Fp",Int_255,Int_255,_psp,*p)   ;
    _spawnl(_P_WAIT,"zad54b","12","345",NULL);
}

#include <stdio.h>
#include <string.h>
#include <graph.h>
#include <stdlib.h>
#include <dos.h>
void ( __interrupt __far *pINT_255)();
int main( int argc, char *argv[], char *envp[])
{
 unsigned  char __far *p;
 char Str[50];
 int i;
    pINT_255 = _dos_getvect((unsigned)0xFF);
    p =(char __far * )(*pINT_255);
    _settextwindow(5,10,20,70);
    _setbkcolor(1);
    _clearscreen(_GWINDOW);
    sprintf (Str,"-- %Fp  ---- %Fp      ",p,*pINT_255);
    _outtext(Str);
    _settextposition(2,1);
    for( i = 0; i < 256; i++)
    {
      sprintf(Str,"%3X",*(p+i));
     _outtext(Str);
     }
     getchar();
     return 0;
  }
```