Б1. Вход и изход: <stdio.h>

Функциите, типовете и макросите, свързани с входните и изходните данни и дефинирани в <stdio.h>, представляват приблизително една трета от библиотеката.

Потокът представлява изходни данни или направление на данни (вход/изход на данни), които могат да се асоциират с диска или с друго периферно устройство. Библиотеката поддържа текстови потоци и двоични потоци, въпреки че при някои системи (най-вече UNIX) те са идентични. Текстовият поток представлява последователност от редове; всеки ред притежава нула или повече символа и завършва с '\n'. Може да се наложи средата да преобразува текстовия поток от или към някакво друго представяне (например да изобрази '\n' като връщане на каретката и преминаване на нов ред). Двоичният поток представлява последователност от необработени байтове, в които се съхраняват вътрешни данни; те притежават свойството записаната в тях информация да остане непроменена, ако се прочете обратно на същата система.

Всеки поток се свързва с някакъв файл или дадено устройство като бъде отворен; връзката се прекратява със затварянето на потока. Отварянето на един файл връща указател към обект от тип FILE, в който е записана необходимата информация за управление на потока. Когато не се появява двусмислие, ще използваме "указател към файл" и "поток" като два взаимозаменяеми термина.

Когато започне изпълнението на една програма, се отварят трите потока - stdin, stdout и stderr.

Б1.1 Операции с файлове

Следните функции боравят с операции, свързани с файлове. Типът size_t представлява беззнаков интегрален (базов) тип, който се явява като резултат от оператора sizeof.

```
FILE *fopen(const char *filename, const char *mode)
```

fopen отваря именувания файл и връща някакъв поток. Ако файлът не може да бъде отворен, fopen връща NULL. Позволените стойности за mode са:

"r" отваря текстов файл за четене
"w" създава текстов файл за писане; ако във файла е имало някакво съдържание, то се пренебрегва
"a" добавя; отваря или създава текстов файл, като пише в края на файла
"r+" отваря текстов файл за актуализация (т.е. за четене и писане)
"w+" създава текстов файл за актуализация; ако във файла е имало някакво съдържание, то се пренебрегва
"a+" добавя; отваря или създава текстов файл за актуализация, като пише в края на файла

Режимите на актуализация позволяват четене и писане в един и същ файл; между четенето и писането (или обратно) може да бъде извикана функцията fflush или някоя функция за позициониране във файла. Ако след първата буква режимът включва b, например "rb" или "w+b", това показва, че ще работим с двоичен файл. Имената на файловете са ограничени до FILENAME_ MAX символа. Могат да бъдат отворени едновременно най-много FOPEN_MAX файла.

```
FILE *freopen(const char *filename, const char *mode, FILE
*stream)
```

freopen отваря файл със зададен режим и го свързва с някакъв поток. Функцията връща stream или NULL, в случай че възникне някаква грешка, f reopen обикновено се използва за промяна на файловете, свързани със stdin, stdout и stderr.

```
int fflush(FILE *stream)
```
При поток на изхода fflush предизвиква запис на всички буферирани, но незаписани данни; при поток на входа ефектът не е дефиниран. Функцията връща EOF, когато възникне грешка при писането; в противен случай връща нула. fflush (NULL) подравнява всички изходни потоци (т.е. прехвърля всички данни от файловите буфери в дисковите файлове).

```
int fclose(FILE *stream)
```
fclose подравнява цялата незаписана информация в stream, пренебрегва непрочетения буфериран вход, освобождава автоматично заделения буфер и след това затваря потока. Функцията връща EOF, ако възникне някаква грешка; в противен случай връща нула.

```
int remove(const char *filename)
```
remove премахва именувания файл. Ако се опитате да получите достъп до файла, след като сте използвали remove, няма да успеете. Ако не успее да премахне файла, функцията връща ненулева стойност.

```
int rename(const char *oldname, const char *newname)
```
rename променя името на файла; ако не успее да промени името, функцията връща ненулева стойност.

```
FILE *tmpfile(void)
```
tmpf ile създава временен файл с режим "wb+", който се премахва автоматично при затваряне или при нормално излизане от програмата, tmpf ile връща поток или NULL, ако не успее да създаде файла.

```
char *tmpnam(char s[L_tmpnam])
```
tmpnam (NULL) създава низ, който не съвпада с името на никой от съществуващите вече файлове, и връща указател към вътрешен статичен масив, tmpnam (s) съхранява низа в s и освен това го връща като стойност на функцията; s трябва да побира най-малко L_tmpnam символа. Всеки път, когато бъде извикана, функцията tmpnam генерира различно име; по време на изпълнението на една програма могат да съществуват най-много TMP_MAX различни имена. Обърнете внимание, че tmpnam създава име, а не файл.

```
int setvbuf (FILE *stream, char *buf, int mode, size_t size)
```
setvbuf контролира буферирането на потока; функцията трябва да бъде извикана преди четене, писане или някаква друга операция. Когато mode е _IOFBF, буферирането е пълно; _IOLBF предизвиква буфериране по редове на текстовия файл; _IONBF не предизвиква буфериране. Ако buf не е NULL, той ще се използва като буфер; в противен случай се заделя памет за буфер, size определя размера на буфера. При грешка setvbuf връща ненулева стойност.

```
void setbuf(FILE *stream, char *buf)
```
Ако buf има стойност NULL, буферирането за дадения поток се изключва. В противен случай функцията setbuf е еквивалентна на (void) setvbuf (stream, buf, _IOFBF, BUFSIZ).

Б1.2 Форматиран изход

Функциите от семейството на printf поддържат форматиран обмен (изход).

```
int fprintf(FILE *stream, const char *format, ...)
```

fprintf преобразува и записва изходните данни в stream; действието се управлява от format. Върната стойност представлява броят на написаните символи. При грешка върнатата

стойност е отрицателна.

Форматиращият низ се състои от два типа обекти: обикновени символи, които се копират в изходния поток, и спецификатори (форматни спецификации), всеки от които предизвиква преобразуване и отпечатване на следващия аргумент на fprintf. Всеки спецификатор започва със символа % и завършва с форматиращ символ. Между % и форматиращия символ могат да се поставят, по ред на изброяването:

- Флагове (в произволна последователност), които променят спецификатора:
-, който задава ляво подравняване на преобразувания аргумент в собственото му поле
+, който показва, че числото винаги ще се отпечатва с някакъв знак
интервал: ако първият символ не е знак, той ще бъде предшестван от интервал.
0: използва се при числови преобразувания; задава запълването на празните позиции в широчината на полето с водещи нули.
#, който определя алтернативен изходен формат. При o, първата цифра ще бъде нула. При x или X, Ox или OX ще предшестват ненулев резултат. При e, E, f, g и G изходните данни винаги ще се записват с десетична точка; при g и G, нулите в края на резултата не се премахват.
- Число, задаващо минималната широчина на полето. Преобразуваните аргументи се отпечатват в поле с минимална широчина, определена от зададеното число, или по-широко при необходимост. Ако преобразуваният аргумент е с по-малко символи от широчината на полето, то ще бъде запълнено с празни позиции отляво (или отдясно, ако е зададено ляво подравняване) до края на широчината на полето. Обикновено символът за запълване е шпацията. Ако обаче за запълване на празните позиции е зададен нулев флаг, символът за запълване е 0.
- Точка, която разделя широчината на полето от точността.
- Число, обозначаващо точността, която показва максималния брой символи, които ще бъдат отпечатани от даден низ; броя на цифрите след десетичната точка при преобразуване със спецификатор e, E или f; броя на значещите цифри при преобразуване с g или G или минималния брой цифри на цяло число (добавят се водещи нули, за да се допълни широчината на полето, ако това е нужно).
- Модификатор на дължината - h, 1 или L. "h" обозначава, че съответстващите му аргументи ще бъдат отпечатани като short или unsigned short; "1" показва, че съответстващият му аргумент е long или unsigned long; "L" обозначава, че аргументът е от long double тип.

Широчината или точността (или и двете едновременно) могат да бъдат зададени като *. В този случай стойността се изчислява чрез преобразуване на следващия аргумент(или аргументи), който трябва задължително да бъде от тип int.

Символите за преобразуване и техните значения са дадени в таблица Б. 1. Ако символът след знака % не е някой от символите за преобразуване, поведението не е дефинирано.

Таблица Б.1. Преобразувания в printf

Символ	Тип на аргумента; Преобразува се до
d,i	int; десетично число.
o	int; беззнаково осмично число (без нула в началото).
x,X	int; беззнаково шестнадесетично число (без 0x или 0X в началото), използва abcdef при 0x или ABCDEF при 0X.
U	int; беззнаково десетично число.
c	int; един символ след преобразуване до unsigned char.
S	char *; отпечатва символите от един низ, докато не достигне до '\0' или докато не достигне броя на символите, зададен от точността.
f	double; запис на десетично число от вида [-] m.dddddd, където броят на d се

задава от точността (по подразбиране е 6). Когато точността е 0, цифрите след десетичната точка се пренебрегват.

e,E double; запис на десетично число от вида [~] m .dddddde±xx или [-] m.ddddddE±xx , където броят на d се задава от точността (по подразбиране е 6). Когато точността е 0, цифрите след десетичната точка се пренебрегват.

g,G double; използва %e или %E, ако степента е по-малка от -4 или е по-голяма или равна на точността; в противен случай използвайте %f. Крайните нули и последната десетична точка не се отпечатват.

P void *; указател (представянето зависи от реализацията).

n int *; броят на символите, написани до момента чрез извикването на printf, се записват в аргумента. Не се извършва преобразуване на аргументи.

% не се преобразува аргумент; отпечатва %.

```
int printf(const char *format, . . . )
```

printf (. . .) е еквивалентна на fprintf (stdout, ...) .

```
int sprintf(char *s, const char *format, . . . )
```

sprintf е еквивалентна на printf, с единствената разлика, че тя записва изходните данни в низа s, който накрая завършва с '\0'. Низът s трябва да бъде достатъчно голям, за да побере резултата. Върнатата стойност представлява броят на записаните символи, без да се брои '\0'.

```
int vprintf(const char *format, va_list arg)
int vfprintf(FILE *stream, const char *format, va_list arg)
int vsprintf(char *s, const char *format, va_list arg)
```

Функциите vprintf, vfprintf и vsprintf са еквиваленти на съответстващите им printf функции, с единствената разлика, че списъкът с аргументите се заменя с arg, който се инициализира с макроса va_start и понякога се извиква макросът va_arg. За подробности вижте дискусията относно <stdarg.h> в раздел Б7.

Б1.3 Форматиран вход

Функциите от семейството на scanf реализират форматиран вход.

```
int fscanf (FILE *stream, const char *format, . . . )
```

fscanf чете от stream под управлението на format и присвоява преобразуваните стойности на последователните аргументи, всеки от които трябва да бъде указател. Когато броят на спецификаторите във format се изчерпи, функцията връща някаква стойност, fscanf връща EOF, ако е достигнат краят на файла или ако по време на преобразуването е възникнала някаква грешка; в противен случай функцията връща като стойност броя на входните елементи, които са имали успешно съответствие със спецификаторите и са били правилно присвоени.

Форматиращият низ обикновено съдържа спецификатори, които управляват преобразуването на входа. Форматиращият низ може да съдържа:

- Шпации или табулатори, които се пренебрегват.
- Обикновени символи (различни от %), които се очаква да съответстват на следващия символ от входния поток, различен от празно пространство.
- Спецификатори, състоящи се от символа %, опционален символ * за подтискане на присвояването, опционално число, обозначаващо максималната широчина на полето, опционална буква h, l или L, показваща широчината на приемащото поле, и форматиращ символ.

Спецификаторът определя преобразуването на следващото входно поле. Обикновено резултатът се поставя в променливата, към която сочи съответстващият аргумент. Ако

подтискането на присвояването е обозначено с *, например %*s, входното поле се прескача; присвояването не се извършва. Входното поле се дефинира като низ от символи, различни от интервали; то се разпростира или до следващия символ за интервал, или до края на широчината на полето, ако такава е зададена. Това показва, че scanf ще чете между границите на един ред, докато намери своя вход, тъй като новите редове се приемат за символи за интервал. (Символите за интервали са шпацията, табулаторът, новият ред, връщането на каретката, вертикалният табулатор и преминаването на нова страница.)

Символът за преобразуване обозначава интерпретацията на входното поле. Съответстващият аргумент трябва да бъде указател. Символите за преобразуване са показани в таблица Б.2.

Символите за преобразуване d, i, h, o, u и x могат да бъдат предшествани от h, което показва, че в списъка с аргументи има указател към short, а не към int, или от буквата 1, за да се покаже, че в списъка с аргументи има указател към long. По подобен начин символите за преобразуване e, f и g могат да бъдат предшествани от 1, за да се покаже, че в списъка с аргументи има указател към double, а не към float. Освен това тези символи за преобразуване могат да бъдат предшествани от L, за да се обозначи указател към long double.

Таблица Б.2. Преобразувания в scanf

Символ	Входни данни; тип на аргумента
d	десетично цяло число; int *.
i	цяло число; int *. Цялото число може да бъде осмично (с 0 в началото) или шестнадесетечино (с 0x или 0X в началото).
o	осмично цяло число (с или без нула в началото); int *.
u	беззнаково цяло число; unsigned int *.
x	шестнадесетично цяло число (с или без 0x или 0X в началото); int *.
c	символи; char *. Следващите входни символи (по подразбиране 1) се поставят в обозначения масив, докато се достигне броят, зададен като широчина на полето. Не се добавя '\0'. В този случай прескачането на празните пространства се подтиска; за да прочетете следващия символ, различен от празно пространство, използвайте %ls.
s	символен низ (без кавички); char *, сочи към достатъчно голям масив от символи, който ще побере низа и крайния '\0'.
e, f, g	число с плаваща запетая; float *. Входният формат за float числата се състои от опционален знак, низ от цифри, който може да съдържа десетична точка, и опционално поле за експонентата, състоящо се от E или e и следвано от знаково цяло число.
p	стойност на указател, която се отпечатва от printf ("%p"); void *.
n	записва в аргумента броя на прочетените до момента символи; int *. Не се четат входни данни. Преобразуваният брой елементи не се увеличава.
[. . .]	най-дългият низ от входни символи, който не е празен и съответства на набора от символи между квадратните скоби; char *. Добавя се '\0'. []...] включва] в набора от символи.
[^ . . .]	най-дългият низ от входни символи, който не е празен и не съответства на набора от символи между квадратните скоби; char *. Добавя се '\0'. [^]...] включва] в набора от символи.
%	буквален %; не се извършва присвояване.

`int scanf(const char *format, . . . )`
scanf (. . .) е идентична на fscanf (stdin, ...) .

`int sscanf(const char *s, const char *format, . . . )`
sscanf (s, ...) е еквивалентът на scanf (. . .) , с единствената разлика, че входните символи се взимат от низа s.

Б1.4 Входно-изходни функции, свързани със символи

`int fgetc(FILE *stream)`
fgetc връща следващия символ от stream като unsigned char (преобразуван до int) или EOF, ако е достигнат краят на файла или е възникнала някаква грешка.

`char *fgets(char *s, int n, FILE *stream)`
fgets прочита най-много n-1 символа и ги поставя в масива s; четенето се прекратява, когато функцията достигне до нов ред. Новият ред се включва в масива и след него се поставя '\0'. fgets връща s или NULL, ако е достигнат краят на файла или е възникнала някаква грешка.

`int fputc(int c, FILE *stream)`
fputc записва в stream символа c (преобразуван до unsigned char). Функцията връща броя на написаните символи или EOF при грешка.

`int fputs(const char *s, FILE *stream)`
fputs записва в stream низа s (който не трябва да съдържа ' \n'). Функцията връща неотрицателна стойност или EOF при грешка.

`int getc(FILE *stream)`
getc е еквивалентна на fgetc, с изключение на факта, че ако е реализирана като макрос, тя може да изчисли stream няколко пъти.

`int getchar(void)`
getchar е еквивалентна на getc (stdin).

`char *gets(char *s)`
gets прочита следващия входен ред и го записва в масива s; функцията заменя крайните нови редове с '\0'. Тя връща s или NULL, ако е достигнат краят на файла или е възникнала някаква грешка.

`int putc(int c, FILE *stream)`
putc е еквивалентна на fputc, с изключение на това, че ако е реализирана като макрос, тя може да изчисли stream няколко пъти.

`int putchar(int c)`
putchar(c) е еквивалентна на putc (c, stdout).

`int puts(const char *s)`
puts записва в stdout низа s и един нов ред. Тя връща EOF, ако е възникнала някаква грешка; в портивен случай връща неотрицателна стойност.

`int ungetc(int c, FILE *stream)`
ungetc връща обратно в stream символа c (преобразуван до unsigned char), където той ще бъде

върнат при следващото четене. За един поток може да се връща само един символ. Не може да се връща EOF. ungetc връща върнатия символ или EOF при грешка.

Б1.5 Функции за директен (небуфериран) вход / изход

`size_t fread(void *ptr, size_t size, size_t nobj, FILE *stream)`
fread чете от stream и записва в масива ptr най-много nobj обекта с големина size, fread връща като стойност броя на прочетените обекти; този брой може да бъде по-малък от първоначално зададения брой. За определяне на статуса трябва да се използват feof и ferror.

`size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *stream)`
fwrite записва nobj на брой обекта с големина size от масива ptr в потока stream. Функцията връща броя на записаните обекти. При грешка този брой е по-малък от nobj.

Б1.6 Функции за позициониране във файл

`int fseek(FILE *stream, long offset, int origin)`
fseek задава позиция във файловия поток stream; следващото четене или писане ще започне от новата позиция. При двоичен файл позицията се задава като отместване с offset символа от origin, като този отстъп може да бъде SEEK_SET (началото), SEEK_CUR (текущата позиция) или SEEK_END (края на файла). При текстов поток offset трябва да бъде 0 или върната от ftell стойност (в този случай origin трябва да има стойност SEEK_SET). При грешка fseek връща ненулева стойност.

`long ftell(FILE *stream)`
ftell връща текущата позиция във файловия поток stream или -1L при грешка.

`void rewind(FILE *stream)`
rewind (fp) е еквивалентна на fseek (fp, 0L, SEEK_SET); clearerr (fp).

`int fgetpos(FILE *stream, fpos_t *ptr)`
fgetpos записва текущата позиция на stream в *ptr, като впоследствие тази позиция може да се използва от fsetpos. Типът fops_t е подходящ за записване на такъв тип стойности. При грешка fgetpos връща ненулева стойност.

`int fsetpos(FILE *stream, const fpos_t *ptr)`
fsetpos позиционира stream на позицията, записана от fgetpos в *ptr. При грешка fsetpos връща ненулева стойност.

Б1.7 Функции за откриване на грешки

Повечето от функциите в библиотеката задават (поддържат) индикатори за статуса на обмена с файла, когато възникне грешка или се достигне краят на файл. Тези индикатори могат да бъдат настроени и проверени. Освен това целочисленият израз errno (деклариран в <errno.h>) може да съдържа номер на грешката, в който да се предоставя допълнителна информация за последната грешка.

`void clearerr(FILE *stream)`
clearerr изчиства индикаторите за край на файл и за грешка за потока stream.

`int feof(FILE *stream)`
feof връща ненулева стойност, ако е зададен индикатор за край на файл за потока stream.

```
int ferror(FILE *stream)
```
terror връща ненулева стойност, ако е зададен индикатор за грешка за потока stream.

```
void perror (const char *s)
```
perror (s) отпечатва s и съобщение за грешка, което зависи от реализацията и което съответства на цялото число в errno. Резултатът е същият като при употребата на

```
fprintf (stderr, "%s: %s\n", s, "съобщение за грешка")
```

Вижте strerror в раздел БЗ.