B1. Ввод-вывод: <stdio.h>

Определенные в **<stdio.h>** функции ввода-вывода, а также типы и макросы составляют приблизительно одну треть библиотеки.

Поток - это источник или получатель данных; его можно связать с диском или с каким-то другим внешним устройством. Библиотека поддерживает два вида потоков: ***текстовый*** и ***бинарный***, хотя на некоторых системах, в частности в UNIXe, они не различаются. *Текстовый поток* - это последовательность строк; каждая строка имеет нуль или более символов и заканчивается символом '\n'. Операционная среда может потребовать коррекции текстового потока (например, перевода '\n' в символы возврат-каретки и перевод-строки).

Бинарный поток - это последовательность непреобразованных байтов, представляющих собой некоторые промежуточные данные, которые обладают тем свойством, что если их записать, а затем прочесть той же системой ввода- вывода, то мы получим информацию, совпадающую с исходной.

Поток соединяется с файлом или устройством посредством его ***открытия***, указанная связь разрывается путем ***закрытия*** потока. Открытие файла возвращает указатель на объект типа *FILE*, который содержит всю информацию, необходимую для управления этим потоком. Если не возникает двусмысленности, мы будем пользоваться терминами "файловый указатель" и "поток" как равнозначными.

Когда программа начинает работу, уже открыты три потока: **stdin**, **stdout** и **stderr**.

B1.1. Операции над файлами

Ниже перечислены функции, оперирующие с файлами. Тип **size_t** - беззнаковый целочисленный тип, используемый для описания результата оператора **sizeof**.

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

fopen открывает файл с заданным именем и возвращает поток или NULL, если попытка открытия оказалась неудачной. Режим *mode* допускает следующие значения:

"r"	- текстовый файл открывается для чтения (от *read* (англ.) - читать);
"w"	- текстовый файл создается для записи; старое содержимое (если оно было) выбрасывается (от *write* (англ.) - писать);
"a"	- текстовый файл открывается или создается для записи в конец файла (от *append* (англ.) - добавлять);
"r+"	- текстовый файл открывается для исправления (т. е. для чтения и для записи);
"w+"	- текстовый файл создается для исправления; старое содержимое (если оно было) выбрасывается;
"a+"	- текстовый файл открывается или создается для исправления уже существующей информации и добавления новой в конец файла.

Режим "исправления" позволяет читать и писать в один и тот же файл; при переходах от операций чтения к операциям записи и обратно должны осуществляться обращения к **fflush** или к функции позиционирования файла. Если указатель режима дополнить буквой *b* (например "rb" или "w+b"), то это будет означать, что файл бинарный. Ограничение на длину имени файла задано константой FILENAME_MAX. Константа FOPEN_MAX ограничивает число одновременно открытых файлов.

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

freopen открывает файл с указанным режимом и связывает его с потоком *stream*. Она возвращает *stream* или, в случае ошибки, NULL. Обычно *freopen* используется для замены файлов, связанных с *stdin*, *stdout* или *stderr*, другими файлами.

`int fflush(FILE *stream);`

Применяемая к потоку вывода функция **fflush** производит дозапись всех оставшихся в буфере (еще не записанных) данных, для потока ввода эта функция не определена. Возвращает EOF в случае возникшей при записи ошибки или нуль в противном случае. Обращение вида *fflush*(NULL) выполняет указанные операции для всех потоков вывода.

`int fclose(FILE *stream);`

fclose производит дозапись еще не записанных буферизованных данных, сбрасывает несчитанный буферизованный ввод, освобождает все автоматически запрошенные буфера, после чего закрывает поток. Возвращает EOF в случае ошибки и нуль в противном случае.

`int remove(const char *filename);`

remove удаляет файл с указанным именем; последующая попытка открыть файл с этим именем вызовет ошибку. Возвращает ненулевое значение в случае неудачной попытки.

`int rename(const char *oldname, const char *newname);`

rename заменяет имя файла; возвращает ненулевое значение в случае, если попытка изменить имя оказалась неудачной. Первый параметр задает старое имя, второй - новое.

`FILE *tmpfile(void);`

tmpfile создает временный файл с режимом доступа "wb+", который автоматически удаляется при его закрытии или обычном завершении программой своей работы. Эта функция возвращает поток или, если не смогла создать файл, NULL.

`char *tmpnam(char s[L_tmpnam]);`

tmpnam(NULL) создает строку, не совпадающую ни с одним из имен существующих файлов, и возвращает указатель на внутренний статический массив. *tmpnam(s)* запоминает строку в *s* и возвращает ее в качестве значения функции; длина *s* должна быть не менее **L_tmpnam**. При каждом вызове *tmpnam* генерируется новое имя; при этом гарантируется не более TMPMAX различных имен за один сеанс работы программы. Заметим, что *tmpnam* создает имя, а не файл.

`int setvbuf(FILE *stream, char *buf, int mode, size_t size);`

setvbuf управляет буферизацией потока; к ней следует обращаться прежде, чем будет выполняться чтение, запись или какая-либо другая операция, *mode* со значением **_IOFBF** вызывает полную буферизацию, с **_IOLBF** - "построчную" буферизацию текстового файла, а *mode* со значением **_IONBF** отменяет всякую буферизацию. Если параметр *buf* не есть NULL, то его значение - указатель на буфер, в противном случае под буфер будет запрашиваться память. Параметр *size* задает размер буфера. Функция *setvbuf* в случае ошибки выдает ненулевое значение.

`void setbuf(FILE *stream, char *buf);`

Если *buf* есть NULL, то для потока *stream* буферизация выключается. В противном случае вызов **setbuf** приведет к тем же действиям, что и вызов (void) setvbuf (stream, buf, _IOFBF, BUFSIZ).

B1.2. Форматный вывод

Функции **printf** осуществляют вывод информации по формату.

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

fprintf преобразует и пишет вывод в поток *stream* под управлением *format*. Возвращаемое значение - число записанных символов или, в случае ошибки, отрицательное значение.

Форматная строка содержит два вида объектов: *обычные символы*, копируемые в выводной поток, и *спецификации преобразования*, которые вызывают преобразование и печать остальных аргументов в том порядке, как они перечислены. Каждая спецификация преобразования начинается с **%** и заканчивается символом-спецификатором преобразования. Между % и символом- спецификатором в порядке, в котором они здесь перечислены, могут быть расположены следующие элементы информации:

- Флаги (в любом порядке), модифицирующие спецификацию:

-	- указывает на то, что преобразованный аргумент должен быть прижат к левому краю поля;
+	- предписывает печатать число всегда со знаком;
пробел	- если первый символ - не знак, то числу должен предшествовать пробел;
0	- указывает, что числа должны дополняться слева нулями до всей ширины поля;
#	- указывает на одну из следующих форм вывода: для *o* первой цифрой должен быть 0; для *x* или *X* ненулевому результату должны предшествовать 0x или 0X; для *e*, *E*, *f*, *g* и *G* вывод должен обязательно содержать десятичную точку; для *g* и *G* завершающие нули не отбрасываются.

- Число, специфицирующее минимальную ширину поля. Преобразованный аргумент будет напечатан в поле, размер которого не меньше указанной ширины, а если потребуется, в поле большего размера. Если число символов преобразованного аргумента меньше ширины поля, то поле будет дополнено слева (или справа, если число прижимается к левому краю). Обычно поле дополняется пробелами (или нулями, если присутствует флаг дополнения нулями).
- Точка, отделяющая указатель ширины поля от указателя точности.
- Число, задающее точность, которое специфицирует максимальное количество символов, печатаемых из строки, или количество цифр после десятичной точки в преобразованиях e, E или f, или количество значащих цифр для g или G - преобразования, или минимальное количество цифр при печати целого (до необходимой ширины поля число дополняется слева нулями).
- Модификаторы **h**, **l** (буква ell) или **L**. "h" указывает на то, что соответствующий аргумент должен печататься как *short* или *unsigned short*; "l" сообщает, что аргумент имеет тип *long* или *unsigned long*; "L" информирует, что аргумент принадлежит типу *long double*.

Ширина, или точность, или обе эти характеристики могут быть специфицированы с помощью *; в этом случае необходимое число "извлекается" из следующего аргумента, который должен иметь тип *int* (в случае двух звездочек используются два аргумента).

Символы-спецификаторы и разъяснение их смысла приведены в таблице B-1. Если за **%** нет правильного символа-спецификатора, результат не определен.

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

printf(...) полностью эквивалентна *fprintf*(stdout, ...).

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

sprintf действует так же, как и *printf*, только вывод осуществляет в строку *s*, завершая ее символом '\0'. Строка *s* должна быть достаточно большой, чтобы вмещать результат вывода. Возвращает количество записанных символов, в число которых символ '\0' не входит.

```
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)
```

Функции **vprintf**, **vfprintf** и **vsprintf** эквивалентны соответствующим *printf*- функциям с той лишь разницей, что переменный список аргументов представлен параметром *arg*, инициализированным макросом *va_start* и, возможно, вызовами *va_arg* (см. в B7 описание **<stdarg.h>**).

Таблица B-1. Преобразования *printf*

Символ	Тип аргумента; вид печати
d, **i**	**int**; знаковая десятичная запись
o	**unsigned int**; беззнаковая восьмеричная запись (без 0 слева)
x, **X**	**unsigned int**; беззнаковая шестнадцатеричная запись (без 0x или 0X слева), в качестве цифр от 10 до 15 используются abcdef для x и ABCDEF для X
u	**unsigned int**; беззнаковое десятичное целое
c	**int**; единичный символ после преобразования в *unsigned char*
s	**char ***; символы строки печатаются, пока не встретится '\0' или не исчерпается количество символов, указанное точностью
f	**double**; десятичная запись вида [-]mmm.ddd, где количество *d* специфицируется точностью. По умолчанию точность равна 6; нулевая точность подавляет печать десятичной точки
e, **E**	**double**; десятичная запись вида [-]m.dddddde±xx или запись вида [-]m.ddddddE±xx, где количество *d* специфицируется точностью. По умолчанию точность равна 6; нулевая точность подавляет печать десятичной точки
g, **G**	**double**; используется **%e** и **%E**, если порядок меньше -4 или больше или равен точности; в противном случае используется **%f**. Завершающие нули и точка в конце не печатаются
p	**void ***; печатает в виде указателя (представление зависит от реализации
n	**int ***; число символов, напечатанных к данному моменту данным вызовом *printf*, *записывается* в аргумент. Никакие другие аргументы не преобразуются
%	никакие аргументы не преобразуются; печатается **%**

B1.3. Форматный ввод

Функции **scanf** имеют дело с форматным преобразованием при вводе

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

fscanf читает данные из потока *stream* под управлением *format* и преобразованные величины присваивает по порядку аргументам, каждый из которых должен быть указателем. Завершает работу, если исчерпался формат. Выдает EOF по исчерпании файла или перед любым преобразованием, если возникла ошибка; в остальных случаях функция возвращает количество преобразованных и введенных элементов.

Форматная строка обычно содержит спецификации преобразования, которые используются для управления вводом. В форматную строку могут входить:

- пробелы и табуляции, которые игнорируются;
- обычные символы (кроме **%**), которые ожидаются в потоке ввода среди символов, отличных от символов-разделителей;
- спецификации преобразования, состоящие из **%**; необязательного знака *****, подавляющего присваивание; необязательного числа, специфицирующего максимальную ширину поля; необязательных **h**, **l** или **L**, указывающих размер присваиваемого значения, и символа-спецификатора преобразования.

Спецификация преобразования определяет преобразование следующего поля ввода. Обычно результат размещается в переменной, на которую указывает соответствующий аргумент. Однако если присваивание подавляется с помощью знака *****, как, например, в %*s, то поле ввода просто пропускается, и никакого присваивания не происходят. Поле ввода определяется как строка символов, отличных от символов-разделителей; при этом ввод строки прекращается при выполнении любого из двух условий: если встретился символ-разделитель или если ширина поля (в случае, когда она указана) исчерпана. Из этого следует, что при переходе к следующему полю *scanf* может "перешагивать" через границы строк, поскольку символ новой строки является символом-разделителем. (Под символами-разделителями понимаются символы пробела, табуляции, новой строки, возврата каретки, вертикальной табуляции и смены страницы.)

Символ-спецификатор указывает на способ интерпретации поля ввода. Соответствующий аргумент должен быть указателем. Список допустимых символов-спецификаторов приводится в таблице B-2.

Символам-спецификаторам **d**, **i**, **n**, **o**, **u** и **x** может предшествовать **h**, если аргумент есть указатель на *short* (а не *int*) или **l** (буква ell), если аргумент есть указатель на *long*. Символам-спецификаторам **e**, **f** и **g** может предшествовать **l**, если аргумент - указатель на *double* (а не *float*), или **L**, если аргумент - указатель на *long double*.

```
int scanf (const char *format, ...);
```

scanf(...) делает то же, что и *fscanf*(stdin, ...).

```
int sscanf (const char *s, const char *format, ...);
```

sscanf(s, ...) делает то же, что и *scanf*(...), только ввод символов осуществляет из строки *s*.

Таблица B-2. Преобразования *scanf*

Символ	Данные на вводе; тип аргумента
d	десятичное целое; **int ***
i	целое: **int ***. Целое может быть восьмеричным (с нулем слева) или шестнадцатеричным (с 0x или 0X слева)
o	восьмеричное целое (с нулем слева или без него); **int ***
u	беззнаковое десятичное целое; **unsigned int ***
x	шестнадцатеричное целое (с 0x или 0X слева или без них): **int ***
c	символы, **char ***. Символы ввода размещаются в указанном массиве в количестве, заданном шириной поля; по умолчанию это количество равно 1. Символ'\0' не добавляется. Символы-разделители здесь рассматриваются как обычные символы и поступают в аргумент. Чтобы прочесть следующий символ-разделитель, используйте %1s
s	строка символов, отличных от символов-разделителей (записывается без кавычек); **char ***, указывающий на массив размера достаточного, чтобы вместить строку и добавляемый к ней символ '\0'
e, **f**, **g**	число с плавающей точкой; **float ***. Формат ввода для *float* состоит из необязательного знака, строки цифр, возможно с десятичной точкой, и необязательного порядка, состоящего из **E** или **e** и целого, возможно со знаком
p	значение указателя в виде, в котором printf ("%p") его напечатает; **void ***
n	записывает в аргумент число символов, прочитанных к этому моменту в этом вызове; **int ***. Никакого чтения ввода не происходит. Счетчик числа введенных элементов не увеличивается
[...]	выбирает из ввода самую длинную непустую строку, состоящую из символов, заданных в квадратных скобках: **char ***. В конец строки добавляется '\0'. Спецификатор вида **[]...]** включает **]** в задаваемое множество символов
[^...]	выбирает из ввода самую длинную непустую строку, состоящую из символов, не входящих в заданное в скобках множество. В конец добавляется '\0'. Спецификатор вида **[^]...]** включает **]** в задаваемое множество символов
%	обычный символ **%**; присваивание не делается

B1.4. Функции ввода-вывода символов

`int fgetc(FILE *stream);`

fgetc возвращает следующий символ из потока *stream* в виде *unsigned char* (переведенную в *int*) или EOF, если исчерпан файл или обнаружена ошибка.

`char *fgets(char *s, int n, FILE *stream);`

fgets читает не более *n-1* символов в массив *s*, прекращая чтение, если встретился символ новой строки, который включается в массив; кроме того, записывает в массив '\0'. Функция *fgets* возвращает *s* или, если исчерпан файл или обнаружена ошибка, NULL.

`int fputc(int c, FILE *stream);`

fputc пишет символ *c* (переведенный в *unsigned char*) в *stream*. Возвращает записанный символ или EOF в случае ошибки.

`int fputs(const char *s, FILE *stream);`

fputs пишет строку *s* (которая может не иметь '\n') в *stream*; возвращает неотрицательное целое или EOF в случае ошибки.

```
int getc(FILE *stream);
```

getc делает то же, что и *fgetc*, но в отличие от последней, если она - макрос, *stream* может браться более одного раза.

```
int getchar(void);
```

getchar() делает то же, что *getc*(stdin).

```
char *gets(char *s);
```

gets читает следующую строку ввода в массив *s*, заменяя символ новой строки на '\0'. Возвращает *s* или, если исчерпан файл или обнаружена ошибка, NULL.

```
int putc(int c, FILE *stream);
```

putc делает то же, что и *fputc*, но в отличие от последней, если *putc* - макрос, значение *stream* может браться более одного раза.

```
int putchar(int c);
```

putchar(c) делает тоже, что *putc*(c, stdout).

```
int puts(const char *s);
```

puts пишет строку *s* и символ новой строки в *stdout*. Возвращает EOF в случае ошибки, или неотрицательное значение, если запись прошла нормально.

```
int ungetc(int c, FILE *stream);
```

ungetc отправляет символ *c* (переведенный в *unsigned char*) обратно в *stream*; при следующем чтении из *stream* он будет получен снова. Для каждого потока вернуть можно не более одного символа. Нельзя возвращать EOF. В качестве результата *ungetc* выдает отправленный назад символ или, в случае ошибки, EOF.

B1.5. Функции прямого ввода-вывода

```
size_t fread(void *ptr, size_t size, size_t nobj, FILE *stream);
```

fread читает из потока *stream* в массив *ptr* не более *nobj* объектов размера *size*. Она возвращает количество прочитанных объектов, которое может быть меньше заявленного. Для индикации состояния после чтения следует использовать *feof* и *ferror*.

```
size_t fwrite(const void *ptr, size_t size, size_t nobj,
              FILE *stream);
```

fwrite пишет из массива *ptr* в *stream nobj* объектов размера *size*; возвращает число записанных объектов, которое в случае ошибки меньше *nobj*.

B1.6. Функции позиционирования файла

```
int fseek(FILE *stream, long offset, int origin);
```

fseek устанавливает позицию для *stream*; последующее чтение или запись будет производиться с этой позиции. В случае бинарного файла позиция устанавливается со смещением *offset* - относительно начала, если *origin* равен **SEEK_SET**; относительно текущей позиции, если *origin* равен **SEEK_CUR**; и относительно конца файла, если *origin* равен **SEEK_END**. Для текстового файла *offset* должен быть нулем или значением, полученным с помощью вызова функции *ftell*. При работе с текстовым файлом *origin* всегда должен быть равен **SEEK_SET**.

```
long ftell(FILE *stream);
```

ftell возвращает текущую позицию потока *stream* или -1L, в случае ошибки.

```
void rewind(FILE *stream);
```

rewind(fp) делает то же, что и fseek(fp, 0L, SEEK_SET); clearerr(fp).

```
int fgetpos(FILE *stream, fpos_t *ptr);
```

fgetpos записывает текущую позицию потока *stream* в **ptr* для последующего использования ее в *fsetpos*. Тип **fpos_t** позволяет хранить такого рода значения, В случае ошибки *fgetpos* возвращает ненулевое значение.

```
int fsetpos(FILE *stream, const fpos_t *ptr);
```

fsetpos устанавливает позицию в *stream*, читая ее из **ptr*, куда она была записана ранее с помощью *fgetpos*. В случае ошибки *fsetpos* возвращает ненулевое значение.

B1.7. Функции обработки ошибок

Многие функции библиотеки в случае ошибки или конца файла устанавливают индикаторы состояния. Эти индикаторы можно проверять и изменять. Кроме того, целое выражение **errno** (объявленное в **<errno.h>**) может содержать номер ошибки, который дает дополнительную информацию о последней из обнаруженных ошибок.

```
void clearerr(FILE *stream);
```

clearerr очищает индикаторы конца файла и ошибки потока *stream*.

```
int feof(FILE *stream);
```

feof возвращает ненулевое значение, если для потока *stream* установлен индикатор конца файла.

```
int ferror(FILE *stream);
```

ferror возвращает ненулевое значение, если для потока *stream* установлен индикатор ошибки.

```
void perror(const char *s);
```

perror(s) печатает *s* и зависимое от реализации сообщение об ошибке, соответствующее целому значению в *errno*, т. е. делает то же, что и обращение к функции *fprintf* вида

```
fprintf(stderr, "%s: %s\n", s, "сообщение об ошибке")
```

См. *strerror* в параграфе B3.