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, ...);

...

B1.3. Форматный ввод

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

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

...

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.