# Файловый ввод и вывод. Форматированный ввод и вывод. Примеры.

**Файл** — это именованный раздел хранилища, обычно расположенный на HDD или SSD. В языке Си **файл** рассматривается как непрерывная последовательность байтов, каждый из которых может быть прочитан индивидуально. В Си предлагаются два способа представления файлов: **текстовый режим** и **двоичный** (*бинарный*) **режим**.

Если в файле *двоичные коды* символов (к примеру, *ASCII* или *Unicode*) используются главным образом для представления текста, почти как в строках *Cu-style*, то такой файл является **текстовым**.
Если двоичные значения в файле представляют код на *машинном языке*, числовые данные , кодировку изображения или музыкального произведения, то содержимое будет *двоичным*.
Различают два уровня файлового ввода/вывода:

- *Низкоуровневый ввод/вывод* предусматривает использование основных служб ввода/вывода, предоставляемых операционной системой.
- *Высокоуровневый ввод/вывод* предполагает применение **стандартного пакета** библиотечных функций Си и определений из заголовочного файла *stdio.h.*
  Стандарт Си поддерживает только **стандартный пакет** ввода- вывода, т.к. нет никакой возможности гарантировать, что все операционные системы могут быть представлены одинаковой низкоуровневой моделью ввода-вывода.
  ***Стандартный ввод -*** обеспечивает ввод данных с клавиатуры Это файл, который читается с помощью функций ***getchar(), scanf()*** и подобными.
  ***Стандартный вывод*** - место, куда направляется обычный вывод программы. Он используется функциями ***putchar(), puts(), printf()*** и подобными.
  ***Стандартный вывод ошибок*** – предназначен чтобы предоставить логически обособленное место для отправки сообщений об ошибках.

Функции ***fopen()*** открывается файл. Эта функция объявлена в заголовочном файле ***stdio.h.***
Ее первым аргументом является ***имя файла*** – это адрес строки, содержащей имя файла. Второй аргумент — строка, ***идентифицирующая режим***, в котором файл должен быть открыт. После успешного открытия файла функция ***fopen()*** возвращает указатель файла, который затем другие функции ввода/вывода могут использовать для указания этого файла.

```
FILE * fp;
fp = fopen("wacky.txt", "r");
```

| Строка режима | Описание |
|---|---|
| "r" | Открыть текстовый файл для чтения |
| "w" | Открыть текстовый файл для записи с усечением существующего файла до нулевой длины или созданием файла, если он не существует |
| "a" | Открыть текстовый файл для записи с добавлением данных в конец существующего файла или созданием файла, если он не существует |
| "r+" | Открыть текстовый файл для обновления (т.е. для чтения и записи) |
| "w+" | Открыть текстовый файл для обновления (чтения и записи), предварительно выполнив усечение файла до нулевой длины, если он существует, или создав файл, если его нет |
| "a+" | Открыть текстовый файл для обновления (чтения и записи) с добавлением данных в конец существующего файла или созданием файла, если он не существует; читать можно весь файл, но записывать допускается только в конец файла |
| "rb", "wb", "ab", ab+", "a+b", "wb+", "w+b", "ab+", "a+b" | Подобны предыдущим режимам, за исключением того, что вместо текстового режима они используют двоичный режим |
| "wx", "wbx", "w+x", "wb+x" или "w+bx" | (C11) Подобны режимам без буквы x, за исключением того, что они отказываются работать, если файл существует, и открывают файл в монопольном режиме, если это возможно |

FILE - производный тип, определенный в ***stdio.h***
Функция **fopen()** возвращает нулевой указатель, если ей не удается открыть файл.
Функции **getc()** и **putc()** работают очень похоже на **getchar()** и **putchar()**. Отличие заключается в том, что этим новым функциям потребуется указать, с каким файлом работать.

```
char ch = getc(fp); //извлечение символа из файла
putc(ch, fp); //Запись символа в файл
```

Функция **getc()** возвращает специальное значение EOF, если она пытается прочитать символ и обнаруживает, что достигнут конец файла.

Функция **fclose(fp)** закрывает файл, идентифицируемый **fp**, при необходимости сбрасывая буферы. В более ответственной программе вы должны удостовериться, что файл закрыт успешно. Функция **fclose()** возвращает значение 0, если файл был закрыт успешно, и **EOF** если нет.

```
if (fclose(fp) != 0)
    printf("Ошибка при закрытии файла %s\n", argv[1]);
```

Функции ***файлового ввода/вывода*** **fprintf()** и **fscanf()** работают аналогично **printf()** и **scanf()**, отличаясь только наличием дополнительного первого аргумента, в котором идентифицируется подходящий файл.

```
int b;
fscanf(stdin, "%d", &b);
fprintf(stdout, "%d", b);
```

Функция **fgets()** читает входные данные до появления первого символа новой строки ("\n") до тех пор, пока не будет прочитано количество символов, на единицу меньше верхнего предела, либо пока не будет обнаружен конец файла, затем fgets() добавляет завершающий нулевой символ, что бы сформировать строку.

```
fgets(buffer, sizeof(buffer), stdin);
```

Функция **fputs()** принимает два аргумента: адрес строки и указатель файла. Она записывает строку, находящуюся в указанной ячейке, в заданный файл. В отличие от **puts()**, функция **fputs()** при выводе не добавляет символ новой строки.

```
fputs(buffer, stdout);
```

Функция **fseek()** позволяет трактовать файл подобно массиву и переходить непосредственно к любому байту в файле, открытом с помощью **fopen()**.

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

*SEEK_SET*: начало файла.
*SEEK_CUR*: текущая позиция указателя.
*SEEK_END*: конец файла.

Функция **ftell()** возвращает текущую позицию в файле как значение **long**.

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

Пример:

```
// Получаем текущую позицию указателя
long pos = ftell(file);
// Перемещаем указатель в начало файла
fseek(file, 0, _SEEK_SET_);
```

```
// Перемещаем указатель на 7-й байт
fseek(file, 7, _SEEK_SET_);
```

Проблема **fseek** и **ftell** в том, что они ограничивают размеры файлов значениями, которые могут быть представлены типом **long**. Вместо этого созданы функции которые работают с новым специализированным форматом ***fpos_t***.
Вызов **fgetpos()** помещает текущее значение типа *fpos_t* в ячейку, указанную *pos*; это значение описывает позицию в файле. Функция возвращает ноль в случае успеха и ненулевое значение при отказе.

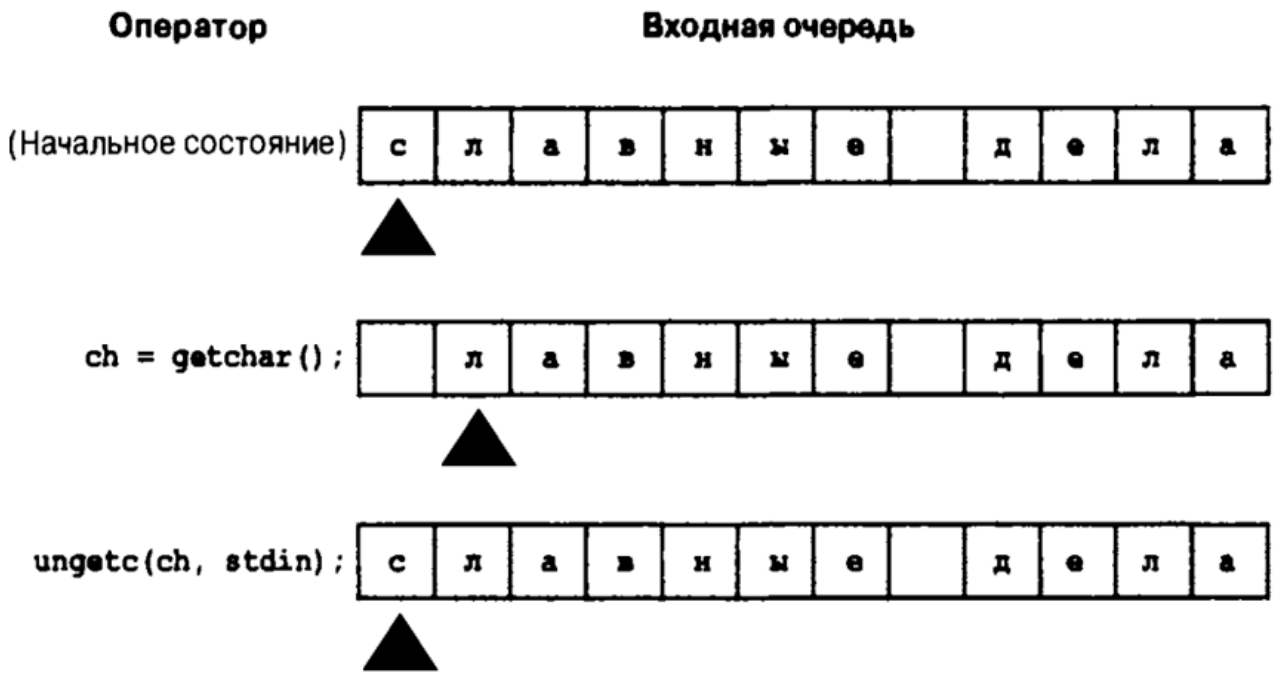
```
int fgetpos(FILE * restrict stream, fpos_t * restrict pos);
```

Вызов **fsetpos()** приводит к использованию значения типа ***fpos_t*** из ячейки, заданной с помощью ***pos***, для установки указателя файла в позицию, которую отражает это значение. Функция возвращает ноль в случае успеха и ненулевое значение при отказе. Значение ***fpos_t*** должно было быть получено предыдущим вызовом **fsetpos()**.

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

Функция **ungetc()** заталкивает символ, указанный в **c**, обратно во входной поток. В случае заталкивания символа во входной поток он будет прочитан следующим вызовом стандартной функции ввода. Если **ungetc()** используется с недопустимым символом (например, **EOF**), поведение не определено.

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




Вызов функции **fflush()** приводит к тому, что любые незаписанные данные в буфере вывода отправляются в выходной файл, идентифицируемый с помощью ***fp***. Этот процесс называется **сбросом буфера**. Если ***fp*** — нулевой указатель, то сбрасываются все буферы вывода.

```
int fflush(FILE *fp);
```

Пример:

```
FILE *file = fopen("output.txt", "w");
if (file == NULL) {
return 1;}
fprintf(file, "Привет, мир!\n");
// Данные записаны в буфер, но еще не сохранены в файле
if (fflush(file) != 0) { fclose(file);
return 1;}
```

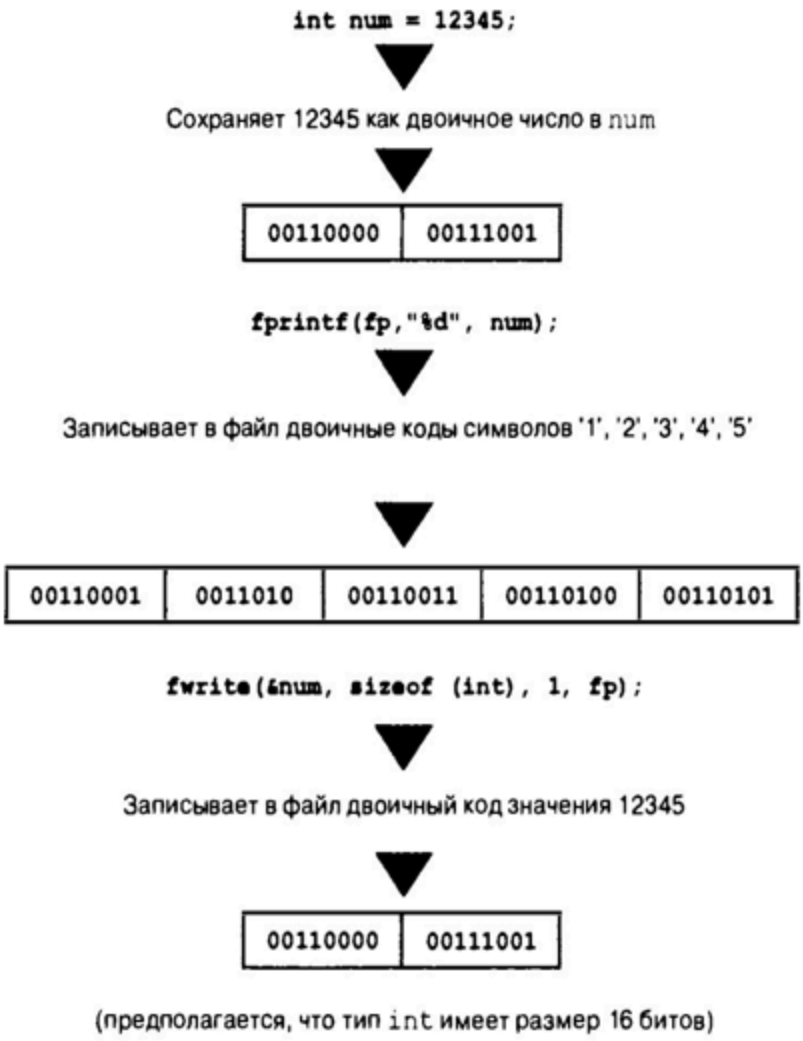
Функция **fwrite()** записывает двоичные данные в файл. Указатель ***ptr*** - это адрес порции данных, предназначенной для записи. Аргумент ***size*** представляет размер в байтах порции данных, подлежащих записи, а ***nmemb*** — количество таких порций.

```
size_t fwrite(const void * restrict ptr, size_t size, size_t nmemb, FILE *
restrict fp);
```

Функция **fread()** принимает такой же набор аргументов, как и **fwrite()**. На этот раз ***ptr*** представляет собой адрес области памяти, куда помещаются данные, прочитанные из

файла, а ***fp*** идентифицирует читаемый файл.

```
size_t fread(void * restrict ptr, size_t size, size_t nmemb, FILE * restrict
fp);
```




Функция **feof()** возвращает ненулевое значение, если при последнем вызове функции ввода был обнаружен конец файла, и ноль в противном случае. Функция **ferror()** возвращает ненулевое значение, если произошла ошибка чтения или записи, и ноль в противном случае.

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