Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №15

Обзор средств ввода-вывода

В программах на языке Си ввод-вывод выполняется с использованием либо стандартной библиотеки ввода-вывода, либо системных вызовов. Системные вызовы в различных операционных системах отличаются и поэтому непереносимы. В отличие от них функции стандартной библиотеки ввода-вывода полностью переносимы.

Для уменьшения количества обращений к устройствам (например, к диску) стандартный ввод-вывод выполняется с буферизацией.

Буфер ввода-вывода — это временная область в основной памяти, которая содержит считываемые или записываемые данные.

Данные пересылаются на устройства (с устройств) большими порциями, размер которых определяется поименованной константой **`BUFSIZ`**.

Поток (stream) — обобщенный термин для обозначения источника или адресата данных, будь то файл или некоторое физическое устройство.

Шаги файлового ввода-вывода

Для файлового ввода-вывода посредством стандартных библиотечных функций ввода-вывода необходимо выполнить следующие шаги:

1. Подключить файл заголовков, используя следующую директиву препроцессора #include <stdio.h>.
2. Объявить указатель на FILE для каждого файла.
3. Открыть файл, используя функцию fopen().
4. Использовать функции чтения-записи стандартного ввода-вывода.
5. Закрыть файл, используя функцию fclose().

В таблице 1 приводятся некоторые поименованные константы из файла **`stdio.h`**. Значения указанных констант зависят от реализации.

Таблица 1

Константа	Смысл константы
`EOF`	признак конца файла
`NULL`	0
`BUFSIZ`	размер буфера ввода-вывода
`FILE`	структура, определенная с помощью **`typedef`**, содержащая информацию об открытом файле

stdin	указатель на **FILE**, открытый для стандартного ввода
stdout	указатель на **FILE**, открытый для стандартного вывода
stderr	указатель на **FILE**, открытый для стандартного вывода сообщений об ошибках

Структура FILE

При включении в программу файла **stdio.h** в системе ОС UNIX выделяется память для массива структур **FILE**. Стандартные библиотечные функции используют структуру **FILE** для хранения информации об открытом фале.

Структура FILE — это структура, в которой хранится информация о том, как открыт файл (например, для чтения), и где позиция следующего чтения или записи. Структура **FILE** инициализируется вызовом функции **fopen()**. Она используется операционной системой и доступна различным функциям для чтения-записи. Программисту не нужно обращаться прямо к полям этой структуры и даже знать, что это за поля. Программист должен лишь объявить указатель на **FILE** перед открытием файла. Этот указатель получает значение во время открытия файла вызовом функции **fopen()** и должен впоследствии передаваться как параметр другим функциям, работающим с данным файлом. Программисту никогда нельзя изменять значение этого указателя. После закрытия файла соответствующая ему переменная файлового указателя на **FILE** может использоваться повторно.

Пример 1

```
/*Образец определения типа  FILE*/
typedef struct
{
  int          _cnt;          /*количество оставшихся в буфере символов*/
  unsigned char *_ptr;        /*следующая позиция в буфере*/
  unsigned char *_base;       /*размещение буфера*/
  unsigned char _flag;        /*признак ошибки чтения-записи или конца файла*/
  unsigned char _file;        /*дескриптор файла ситемы UNIX*/
} FILE;
```

Буферизованный ввод-вывод

Буферизация - способ организации ввода-вывода в программе, позволяющий минимизировать число обращений к устройству.

При чтении из дискового файла, блок данных копируется с диска в пользовательский буфер. Многими операционными системами используется также промежуточный "системный буфер". Этот блок данных, называемый также физическим блоком или блоком ввода-вывода, имеет объем, заданный поименованной константой **BUFSIZ**, определенной в файле

заголовков **stdio.h**. Операции чтения из файла фактически читают данные из этого буфера. При исчерпании буфера вновь выполняется чтение с диска, и буфер пополняется. Обращение к диску лишь при необходимости обеспечивает существенную экономию времени. Программисты могут читать из файла множеством способов (посимвольно, построчно и т.д.) и не задумываться о минимизации количества обращений к диску. Стандартный пакет ввода-вывода гарантирует решение этой проблемы.

В области данных выполняющейся программы содержится множество структур FILE - по одной для каждого файла. Первые три структуры зарезервированы для потоков stdin, stdout и stderr. Следующая структура обычно используется для первого файла, открытого функцией fopen(). Одно из полей структуры **FILE** указывает на связанный с файлом пользовательский буфер. Еще одно поле указывает позицию буфера, с которой будет работать следующая операция чтения или записи. Остальные поля содержат информацию о том, как открывался файл (**r**, **w**, **a**, **r+**, **w+**, **a+**), был достигнут ли конец файла и т.д.

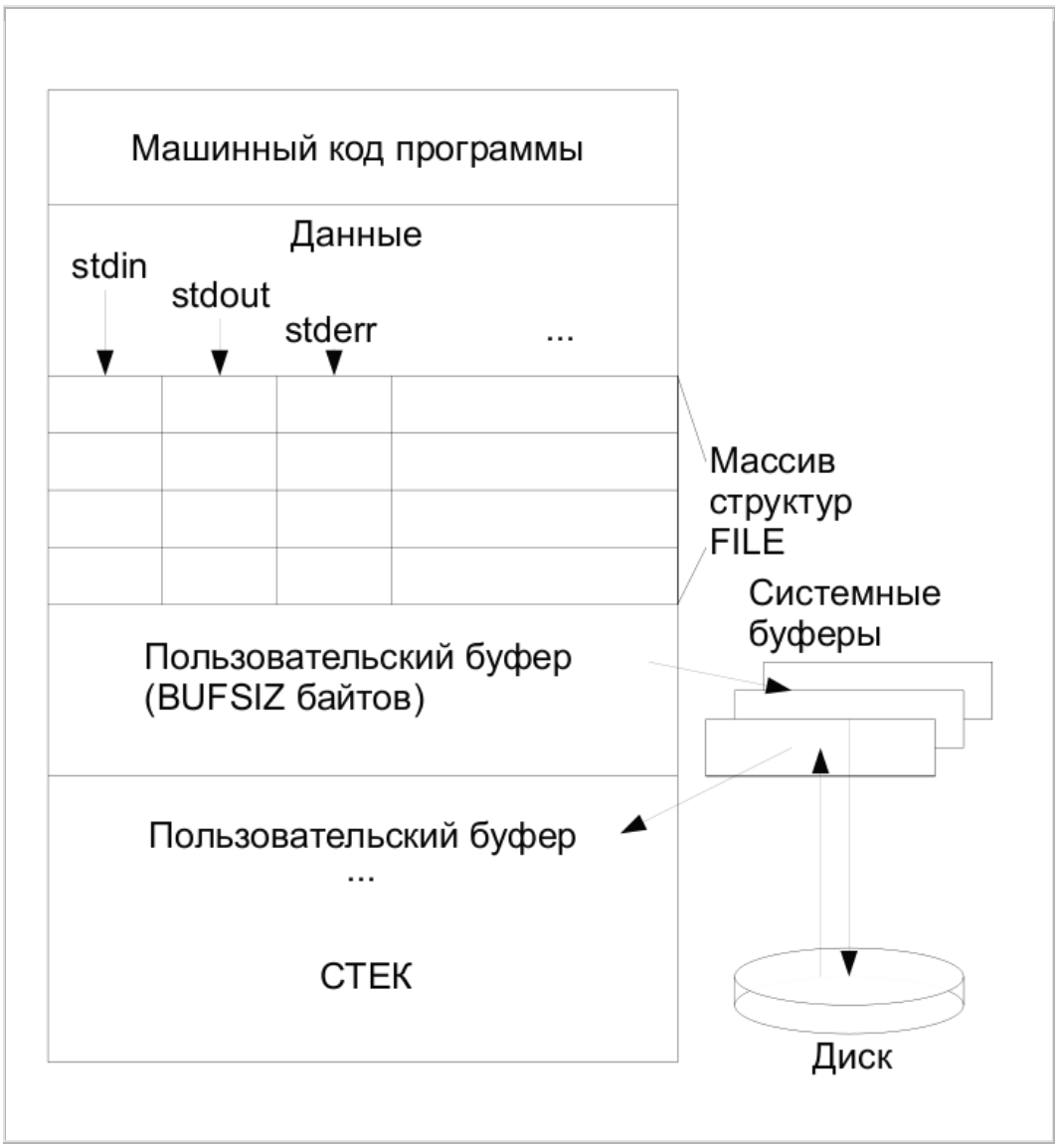

Рис. 1. Буферизованный ввод-вывод

Функция fopen()

Функция fopen() — функция открытия файла.

FILE *fopen(const char *имя_файла, const char *тип)

Первым аргументом функции **fopen()** является адрес строки, содержащей путевое имя файла. Второй аргумент—это адрес строки, определяющей тип работы с открываемым файлом.

Заданный именем файл открывается в соответствии с указанным типом. Тип может принимать следующие значения:

- **"r"** — текстовый файл открывается для чтения (read);
- **"w"**- текстовый файл создается для записи; старое содержимое, если оно было, выбрасывается (write);
- **"a"** — текстовый файл открывается или создается для записи в конец файла (add);
- **"r+"** — текстовый файл открывается для исправления, т.е. для чтения и 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.

Функция **fopen()** возвращает указатель на структуру FILE, который затем присваивается переменной. Этот указатель называют *внутренним*. После открытия файла к нему обращаются с использованием указателя на этот файл, имя файла больше не потребуется.

Для обозначения двоичных файлов к типу может добавляться символ **b** (например, **"rb"**, **"wb"**, **"r+b"**). Однако, ОС UNIX не делает никаких различий между двоичными и текстовыми файлами, следовательно, символ **b** игнорируется.

Объявление прототипа функции **fopen()** находится в файле заголовков **stdio.h**, поэтому программы, использующие эту функцию, должны включать следующую директиву препроцессора:

#include <stdio.h>

Пример 1

```
#include <stdio.h>
int main()
{
  FILE *fp;
  fp = fopen("logfile", "w");
  if (fp==NULL)
     printf("Open failed\n");
  ...
}
```

Для обеспечения уверенности в успешном открытии файла необходимо проверять код, возвращаемый функцией **fopen()**.

В таблице 1 приводятся результаты успешного открытия файла.

Таблица 1

	чтение "r"	**запись "w"**	**добавление "a"**
файл существует	-	старое содержимое отбрасывается	-
файл не существует	ошибка	файл создается	файл создается

Если файл существует и открывается для чтения или дополнения, то он не изменяется при открытии. При открытии существующего файла для записи, он усекается. Если файл, открываемый для записи, не существует, то он будет создан. Однако, открытие несуществующего файла для чтения повлечет ошибку.

Открытие файла с типом **"r+"**, **"w+"** или **"a+"** называют открытием для обновления, что означает, что все три типа допускают как чтение, так и запись. При таком открытии выполняются те же действия с существующими и несуществующими файлами, что и при открытии с соответствующими типами **"r"**, **"w"** и **"a"**. При использовании типов **"a"** и **"a+"** запись будет выполняться в конец файла.

Функция fclose()

Функция fclose() - стандартная функция ввода-вывода языка Си, с помощью которой осуществляется закрытие файла.

```
int fclose (FILE *stream)
```

Функция **fclose()** закрывает файл, на который указывает ее параметр. При закрытии файла его буфер записывается (вытесняется) на диск, если вывод реально выполнялся. Функция **fclose()** возвращает EOF в случае ошибки и нуль в противном случае.

Хотя открытый файл автоматически закрывается при завершении программы, даже если функция **fclose()** не вызвана, рекомендуется закрывать файл явно. Во-первых, при этом освобождается место в памяти, которое может быть использовано для открытия нового файла. Во-вторых, буфер, если он есть, записывается за приемлемое время. Одновременно в программе может быть открыто не более 20 файлов.

Объявление прототипа функции **fclose()** находится в файле заголовков **stdio.h**, поэтому программы, использующие эту функцию, должны включать следующую директиву препроцессора:

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

Пример 1

Эта программа копирует файл, заданный вторым аргументом командной строки в файл, заданный третьим аргументом командной строки.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
FILE *rp, *wp;

if(argc != 3)
  {
   printf ("Имя второго файла не определено\n");
   exit (1);
  }
if((rp=fopen(argv[1],"r"))==NULL)
  {
   printf ("Не открыт файл %s\n",argv[1]);
   exit (2);
  }
if((wp=fopen(argv[2],"w"))==NULL)
  {
  printf ("Не открыт файл %s\n",argv[2]);
  fclose(rp);
  exit (2);
}
/*здесь должны быть функции чтения-записи*/

fclose(rp);
fclose(wp);
exit (0);
}
```

Функции чтения-записи

Стандартная библиотека ввода-вывода содержит множество функций для чтения и записи. Файл может читаться посимвольно, построчно, с форматными преобразованиями или поблочно.

Когда аргументом функции является указатель на FILE, этот указатель в описании функции называют потоком.

Объявления прототипов функций чтения-записи находятся в файле заголовков **stdio.h**, поэтому программы, использующие эти функции, должны включать следующую директиву препроцессора:

#include <stdio.h>

EOF — признак конца файла определен в файле заголовков **stdio.h**. Значение **EOF** зависит от компилятора (типичным значением для ОС

UNIXявляется –1). Значение **EOF** может быть введено с клавиатуры путем нажатия клавиш <CTRL>, <D> в начале строки.

Описания функций чтения-записи

int fgetc(FILE *stream)

Функция возвращает следующий символ из потока в формате **int** или **EOF**, если исчерпан файл или обнаружена ошибка.

int fputc(int c, FILE *stream)

Функция пишет символ, переведенный в формат **unsigned char** в поток, возвращает переданный символ или **EOF** в случае ошибки.

int ungetc(int c, FILE *stream)

Функция отправляет символ, переведенный в формат **unsigned char**, обратно в поток. При следующем чтении из потока он будет получен снова. Для каждого потока вернуть можно не более одного символа. Нельзя возвращать **EOF**. Функция возвращает переданный символ или **EOF** в случае ошибки.

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

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

int fputs(char *s, FILE *stream)

Функция пишет строку **s**, которая может не иметь символ **'\n'**, в поток, возвращает неотрицательное целое или **EOF** в случае ошибки.

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

Функция читает данные из потока под управлением формата и преобразованные величины присваивает аргументам, каждый из которых должен быть указателем. Завершает работу, если исчерпался формат. Возвращает **EOF** по исчерпании файла или перед любым преобразованием, если возникла ошибка. В остальных случаях функция возвращает количество преобразованных и введенных элементов. Правила построения форматной строки такие же, как для функции scanf().

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

Функция преобразует и пишет вывод в поток под управлением формата. Возвращаемое значение — число записанных символов или, в случае ошибки, отрицательное значение. Правила построения форматной строки такие же, как для функции printf().

int fread (void *ptr, size_t size, size_t nitems, FILE *stream)

Функция читает из потока в массив **ptr** не более **nitems** объектов размера **size**. Функция возвращает количество прочитанных объектов, которое может быть меньше заявленного. Для определения состояния после чтения следует использовать функции обработки ошибок **feof()** и **ferror()**.

int fwrite (const void *ptr, size_t size, size_t nitems, FILE *stream)

Функция пишет из массива **ptr** в поток **nitems** объектов размера **size**. Функция возвращает количество записанных объектов, которое в случае ошибки меньше **nitems**.

В следующих примерах рассмотрены различные варианты копирования файла. Предполагается, что файлы предварительно открыты с помощью функции fopen() и после использования будут закрыты функцией fclose().

Пример 1

Копирование файла посимвольно

```
while((c=fgetc(rp)) != EOF)
  fputc(c, wp);
```

Пример 2

Копирование файла построчно

```
char buf[256];
while((fgets(buf, 256, rp)) != NULL)
  fputs(buf, wp);
```

Пример 3

Копирование файла в соответствии с указанным форматом

```
char name[40];
int id;
while(fscanf(rp,"%s %d", name, &id) != EOF)
  fprintf (wp, "%s %d\n", name, id);
```

Пример 4

Копирование файла поструктурно

```
struct info{
        char name[50];
        int id;
        } part;
while (fread(&part, sizeof(part), 1, rp) != 0)
  fwrite (&part, sizeof(part),1, wp);
```

В данных примерах **rp** — указатель FILE, используемый для доступа к файлу, открытому для чтения, **wp** — для записи. Эти указатели инициируются при открытии файлов, содержат информацию об открытии файла и не должны изменяться программистом. Эти указатели совместно используются функциями стандартной библиотеки ввода-вывода так, чтобы гарантировать, что следующие друг за другом вызовы функций чтения или записи обеспечивают последовательный доступ. Поэтому данные указатели иногда называют внутренними указателями.

Функции getchar() и putchar()

Функция getchar() — стандартная функция языка Си для ввода символа.

Объявление прототипа функции **getchar()**:

```
int getchar(void)
```

Функция putchar() — стандартная функция языка Си для вывода символа.

Объявление прототипа функции **putchar()**:

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

Эти объявления находятся в файле заголовков **stdio.h**, поэтому программы, использующие эти функции, должны включать следующую директиву препроцессора:

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

Функция **getchar()** читает из стандартного устройства ввода один символ и возвращает его код в соответствии с таблицей кодирования. Если при попытке чтения были обнаружены конец файла или ошибка, функция **getchar()** возвращает значение EOF. При вводе с терминала **getchar()** возвращает **EOF**, если пользователь нажал в начале строки комбинацию клавиш <CTRL><d>.

Программа, работающая в среде ОС UNIX и использующая функцию **getchar()** для ввода, не получит вводимые с клавиатуры символы до тех пор, пока пользователь не нажал клавишу <RETURN>. Это позволяет пользователю корректировать набираемые символы (например, с помощью забоя) прежде, чем программа их получит. Однако, некоторые программы, например, редактор “vi”, работают с терминалом в режиме, при котором вводимые символы немедленно становятся доступными программе.

Функция **putchar()** выводит один символ в стандартный вывод (по умолчанию — на экран терминала), возвращает при успешном завершении код выводимого символа иначе **EOF**. Обычно возвращаемое значение не обрабатывается.

Пример 1

```
#include <stdio.h>
/* копирование ввода на вывод */
int main()
{
  int c;
```

```
 while ((c=getchar())!=EOF)
   putchar(c);
}
```

Функция ungetc()

Функция ungetc() — стандартная функция языка Си для работы с символом.

Объявление прототипа функции **ungetc()**:

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

Это объявление находится в файле заголовков **stdio.h**, поэтому программы, использующие эту функцию, должны включать следующую директиву препроцессора:

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

Функция **ungetc()** возвращает один символ во входной поток при условии, что выполнялось хотя бы одно чтение. Следующая операция чтения прочитает этот символ.

Часто эта функция реализуется как макрокоманда, определенная в файле заголовков **stdio.h**. Функция **ungetc()** используется для вставки последнего прочитанного символа обратно во входной поток так, что следующая операция чтения прочтет его снова. Это может потребоваться в случае, когда один символ читается много раз, или когда значение прочитанного символа не принадлежит требуемому диапазону (например, ожидается число, а было прочитано 'z').

Вставка символа гарантируется лишь тогда, когда из входного потока было прочитано хотя бы что-нибудь, и когда поток буферизован (открыт функцией fopen()). Если функция **ungetc()** не может вставить символ, она возвращает EOF.

Пример 1

Функция выбрасывает из входного потока символы промежутков.

```
#include <stdio.h>
void skip_whites(FILE *fp)
{
int c;
while ((c=fgetc(fp)) == ' ' || c=='\t' || c== '\n')
  ;
ungetc(c,fp); /*возвращает во входной поток отличный от промежутков символ*/
}
```

Обзор функций форматированного ввода-вывода

В языке программирования Си нет встроенных операторов ввода-вывода. Операции ввода-вывода могут осуществляться с помощью функций из стандартной библиотеки ввода-вывода. К таким функциям относятся и *функции форматированного ввода-вывода*, которые осуществляют

ввод или вывод данных и их преобразования в нужное представление. Эти функции представлены в табл. 1.

Таблица 1

Устройство ввода-вывода	функция ввода	функция вывода
стандартное устройство ввода-вывода	**printf()**	**scanf()**
файл	**fprintf()**	**fscanf()**
строка	**sprintf()**	**sscanf()**

Функция printf() обеспечивает форматированный вывод: программист может задать ширину колонок, выравнивание по левому или правому краю, вывод чисел в десятичной или восьмеричной записи и т.д.

Функция scanf() осуществляет ввод в определенном формате, например, три целых числа, разделенных табуляциями.

Функция **printf()** выводит информацию на стандартное устройство вывода, обычно являющееся монитором терминала. Используя средства системы ОС UNIX, стандартное устройство вывода можно переназначить на вывод в файл, в конвейер (pipe) или в файл устройства.

Функции **fprintf()** и **sprintf()** аналогичны функции **printf()**.Функции **sprintf()** осуществляет вывод в строку, завершая вывод символом '\0' (признак конца строки). Строка должна быть достаточно большой, чтобы вмещать результат вывода.

Функция **scanf()** читает информацию со стандартного устройства ввода, который обычно является клавиатурой терминала. Используя средства системы ОС UNIX, стандартное устройство ввода можно переназначить на ввод из файла, из конвейера или из файла устройства.

Функция printf()

Функция printf() — функция форматированного вывода языка Си, осуществляющая вывод в стандартный поток **stdout** в соответствии с форматом.

int printf(формат[,аргументы]...)

Функция printf() возвращает количество выведенных символов или, в случае ошибки, отрицательное число.

Формат — это строка, заключенная в двойные кавычки, которая может содержать литеральные символы, копируемые в поток вывода, и спецификации преобразования.

Спецификация преобразования — последовательность символов, начинающаяся с символа % и заканчивающаяся символом-спецификатором.

Объявление прототипа функции **printf()** находится в файле заголовков **stdio.h**, поэтому программы, использующие эту функцию, должны включать следующую директиву препроцессора:

#include <stdio.h>

Спецификаторы преобразования должны соответствовать типу указанных аргументов. При несовпадении не будет выдано сообщений об ошибках ни при компиляции, ни при выполнении программы, но при этом вывод программы может содержать "мусор".

Таблица 1

Символ-спецификатор	Тип аргумента; вид печати
c	**int**; единичный символ после преобразования в **unsigned char**
d, i	**int**; знаковое целое число в десятичной записи
o	**int**; знаковое целое число в восьмеричной записи
x, X	**int**; знаковое целое число в шестнадцатеричной (a-f или A-F) записи
u	**int**; беззнаковое целое число в десятичной записи
e, E	**double**; десятичная форма записи вида [-]m.dddddde∓xx или [-]m.ddddddE∓xx, где количество d определяется точностью. По умолчанию точность равна 6, нулевая точность подавляет печать десятичной точки
f	**double**; десятичная форма записи вида [-]mmm.ddd, где количество d определяется точностью. По умолчанию точность равна 6, нулевая точность подавляет печать десятичной точки
g, G	**double**; используется %e или %E, если порядок меньше -4 или больше или равен точности, в противном случае используется %f. Завершающие нули и десятичная точка не печатаются
s	**char**; печатает символы, расположенные до признака конца строки, или в количестве, заданном точностью
p	**void ***;указатель
n	**int ***; число символов, напечатанных к данному моменту данным вызовом функции, записывается в аргумент (никакие другие аргументы не преобразуются)
%	Аргумент не преобразуется; печатается символ %

Пример 1

```
#include <stdio.h>
int main()
{
  char c = 'j';
  int val = 59;
  float total = 7500.5;
  long double var = 63.78;
  char initials[4];
  initials[0] = 'I';
  initials[1] = 'M';
```

```
  initials[2] = 'R';
  initials[3] = '\0';
  printf("%c\n", c);
  printf("%d\n", val);
  printf("%o\n", val);
  printf("%x\n", val);
  printf("%e\n", total);
  printf("%e\n", total);
  printf("%Lf\n", var);
  printf("%s\n", initials);
}
```

вывод:

```
j
59
73
3b
7.500500e+03
7500.500000
63.780000
IMR
```

Дополнительное форматирование

Функция **printf()** предоставляет множество возможностей по форматированию вывода.

В спецификации преобразования между символом % и символом-спецификатором могут быть расположены следующие элементы (в указанном порядке):

- Флаги (в любом порядке), модифицирующие спецификацию:
  - Символ минус (-): выравнивание по левому краю. Задание ширины поля по умолчанию устанавливает выравнивание по правому краю. Для выравнивания по левому краю надо указать символ минус (-) перед шириной поля, например, **%-15d, %-15s**.
  - Символ плюс (+): печать числа со знаком. По умолчанию перед отрицательными числами выводится минус (-). Для того, чтобы перед положительными числами выводился плюс (+), надо использовать **%+d, %+f** и т.д.
  - Символ пробел: если число положительное или нуль, то при печати числу будет предшествовать пробел.
  - Символ нуль (0): указывает, что числа должны дополняться слева нулями до всей ширины поля.
  - Символ #. Для печати ведущих нулей восьмеричных чисел используется спецификатор **%#o**, для вывода 0x перед шестнадцатеричными числами используется спецификатор **%#x**. При использовании **%#f** и **%#e** десятичная точка будет выводиться, даже если в числе нет дробной части.

- Число, определяющее минимальную ширину поля. Минимальная ширина поля используется для задания столбцов. Например, **%15d** напечатает по меньшей мере 15 символов. Число будет выравнено по правому краю и дополнено слева до 15 символов пробелами. При применении этого формата на последующих строчках будет напечатан выравненный по правому краю столбец шириной 15 символов. Число печатается, даже если его длина превышает 15 символов, так как 15 — это минимальная ширина поля.
- Символ точка (.), отделяющий ширину поля от величины, устанавливающей точность.
- Число (точность) использующееся для ограничения количества выводимых символов. При использовании вместе с **%f** (например, **%.2f**), она ограничивает количество цифр после десятичной точки (в данном примере до 2). **%15.2f** выводит вещественное число с двумя знаками после десятичной точки в поле с минимальной шириной 15 символов, включая десятичную точку. Использование точности с **%s** (например, **%.15s**) определяет максимальное количество выводимых символов (в данном примере 15). Точное количество выводимых символов можно контролировать, задавая и минимальную ширину поля и точность (максимальное количество символов), например, 15.15.
- Модификаторы **h**, **l** или **L**. Символ **h** указывает на то, что соответствующий аргумент должен печататься как **short** или **unsigned short**; символ **l** сообщает, что аргумент имеет тип **long** или **unsigned long**; символ **L** сообщает, что аргумент имеет тип **long double**.

Пример 2

Пусть для вывода некоторого набора данных используется следующий вариант функции **printf()**:

```
printf("%d %d %s %f\n", mod, gt, it, cst);
```

Вывод будет выглядеть следующим образом:

```
2901 6 Cerebral Calculator 75.489998
30 7229 Blue Ribbon Cable 26.000000
31650 100 Glow Worm Glare Screen 89.989998
2 677 Personal Mainframe 9000.000000
```

Пример 3

Пусть для вывода этого же набора данных используется вариант функции **printf()** с возможностями дополнительного форматирования:

```
printf("%-6d %4d %-24s %7.2f\n", mod, gt, it, cst);
```

В этом случае вывод будет выглядеть следующим образом:

```
2901      6 Cerebral Calculator         75.48
30     7229 Blue Ribbon Cable           26.00
31650   100 Glow Worm Glare Screen      89.98
2       677 Personal Mainframe        9000.00
```

Функция scanf()

Функция scanf() — функция форматированного ввода языка Си, осуществляющая ввод со стандартного потока **stdin** в соответствии с форматом. При необходимости функция преобразует вводимые символы в числа в соответствии с форматом и сохраняет данные по указанным адресам. Функция прекращает ввод после первой же ошибки, оставляя ошибочный символ непрочитанным. Функция **scanf()** возвращает количество прочитанных и преобразованных полей или признак конца файла (EOF) при достижении конца файла.

Функция **scanf()** — это функция из стандартной библиотеки ввода-вывода.

Объявление прототипа функции **scanf()** находится в файле заголовков **stdio.h**, поэтому программы, использующие эту функцию, должны включать следующую директиву препроцессора:

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

Синтаксис функции **scanf()** аналогичен синтаксису функции printf().

```
int scanf()(формат[,список указателей]...)
```

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

- пробелы и табуляции, которые игнорируются;
- обычные символы (кроме символа %), которые ожидаются в потоке ввода среди символов, отличных от символов-разделителей;
- спецификации преобразования

Для функции **scanf()** спецификация преобразования может состоять из следующих символов:

- символ %
- символ *(необязательный), подавляющий присваивание;
- число(необязательное), специфицирующее максимальную ширину поля;
- символ h или l, или L(необязательный), указывающий размер присваемого значения;
- символ-спецификатор

Допустимые значения символа-спецификатора приведены в таблице 1.

Таблица 1

Символ-спецификатор	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 целое десятичное число; **int ***.
x	Шестнадцатеричное целое (с 0x или 0X слева или без

	них); **int ***.
c	Символ; **char ***. Символы ввода размещаются в указанном массиве символов в количестве, заданном шириной поля, по умолчанию это количество равно 1. Символ '\0' (признак конца строки) не добавляется. Символы-разделители здесь рассматриваются как обычные символы и поступают в аргумент.
s	Строка символов, отличных от символов-разделителей (записывается без двойных кавычек); **char ***, указывающий на массив размера достаточного, чтобы вместить строку и добавленный 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 функция **printf()** его напечатает; **void ***.
n	Записывает в аргумент число символов, прочитанных к данному моменту данным вызовом функции; **int ***. Никакого чтения ввода не происходит. Счетчик числа введенных элементов не увеличивается.
[...]	Выбирает из ввода самую длинную ненулевую строку, состоящую из символов, заданных в квадратных скобках; **char ***. В конце строки добавляется символ **'\0'**. Спецификатор вида []...] включает символ] в заданное множество символов.
[^...]	Выбирает из ввода самую длинную ненулевую строку, состоящую из символов, не входящих в заданное в квадратных скобках множество; **char ***. Спецификатор вида [^]...] включает символ] в заданное множество символов.
%	Обычный символ %; присваивание не делается.

Функция **scanf()** использует адрес аргумента для размещения преобразованных данных. Распространенной ошибкой при использовании функции scanf() является отсутствии операции получения адреса (&). В этом случае функция **scanf()** интерпретирует значение аргумента как адрес и либо записывает данные в непредсказуемое место памяти, либо пытается сделать это в защищенную от записи область, что влечет аварийное завершение работы программы.

Значение, возвращаемое функцией **scanf()**, может использоваться для проверки правильности ввода данных.

Пример 1

```
#include <stdio.h>
int main()
{
```

```
 int number;

 printf("Введите число");
 while(scanf("%d",&number) != 1)
   printf("Ошибка. Попытайтесть еще раз ");
 printf("Спасибо. Вы ввели число %d\n", number);
...
}
```

Экран монотора:

Введите число tyuh

Ошибка. Попытайтесть еще раз Ошибка. Попытайтесть еще раз Ошибка. Попытайтесть еще раз

В прим. 1 в программе сделана неудачная попытка обеспечить ввод того, что нужно программисту. При неудачном вводе числа "ошибочный" символ (в данном случае t) остается во входном потоке. При повторных вызовах функции **scanf()** ввод вновь начинается с этого символа и попытка ввода оказывается безуспешной. Необходимо убрать ненужные символы из входного потока. В прим. 2 показано, как можно очистить входную строку, используя функцию getchar().

Пример 2

```
#include <stdio.h>
int main()
{
 int number;

 printf("Введите число");
 while(scanf("%d",&number) != 1)
   {
    while(getchar() != '\n')
      ;          /* очищает строку*/
    printf("Ошибка. Попытайтесть еще раз ");
   }
 printf("Спасибо. Вы ввели число %d\n", number);
...
}
```

Экран монотора:

Введите число tyuh

Ошибка. Попытайтесть еще раз 34

Спасибо. Вы ввели число 34

Функции gets() и puts()

Функция gets() — стандартная функция языка Си для ввода строки.

Объявление прототипа функции **gets()**:

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

Функция puts() — стандартная функция языка Си для вывода строки.

Объявление прототипа функции **puts()**:

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

Эти объявления находятся в файле заголовков **stdio.h**, поэтому программы, использующие эти функции, должны включать следующую директиву препроцессора:

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

Функция **gets()** вводит входную строку из потока stdin. При этом символ новой строки отбрасывается, а введенные символы дополняются в конце нулевым символом (символом '\0'). Необходимо позаботиться, чтобы принимающий массив символов был достаточно большим для размещения наибольшей возможной строки. Функция возвращает **NULL** по концу файла, если символов для чтения нет.

В отличии от функции scanf(), использующей спецификатор **%s**, функция **gets()** читает полную строку, включая символы пробела и табуляцию.

Функция **puts()** выводит заканчивающуюся нулевым символом строку символов в поток stdout. При этом вместо нулевого символа выводится символ новой строки.

Функции **fgets()** и **fputs()** подобны функциям **gets()** и **puts()**.

Пример 1

```
#include <stdio.h>
#define LINELEN 256
int main()
 {
 char name[LINELEN+1];
 printf("Введите полное имя:");
 if (gets(name) == NULL)
  printf("Ошибка\n");
 else
  puts(name);
}
```

Вывод программы:

```
Введите полное имя:Иванов Петр Васильевич
Иванов Петр Васильевич
```

Вытеснение буфера

Функция fflush() — стандартная функция языка Си для вытеснения содержимого пользовательского буфера в файл или в системный буфер, не дожидаясь заполнения пользовательского буфера.

Объявление прототипа функции **fflush()**:

int fflush(FILE *stream)

Это объявление находится в файле заголовков **stdio.h**, поэтому программы, использующие эту функцию, должны включать следующую директиву препроцессора:

#include <stdio.h>

Функция **fflush()** обычно используется для файлов, открытых для записи, возвращает значение EOF в случае возникшей при записи ошибки или нуль в противном случае.

Эту функцию можно использовать для сброса ввода (остатка входного файла).

Пример 1

```
fflush(stdin); /*не запрашивает уже выполненный ввод*/
printf("ВВедите данные: ");
gets(buf);
```

Произвольный доступ

Если не выполнялись специальные действия, операции чтения и записи обращаются к последовательным байтам в файле. Позиция в файле для следующей операции чтения или записи хранится во внутреннем указателе — поле в структуре FILE, автоматически устанавливаемом и изменяемом стандартными функциями ввода-вывода. Однако можно изменить место следующего чтения или записи, используя функцию **fseek()**, модифицирующую внутренний указатель.

Объявление прототипа функции **fseek()**:

int fseek(FILE *stream, long offset, int ptrname)

Это объявление находится в файле заголовков **stdio.h**, поэтому программы, использующие эту функцию, должны включать следующую директиву препроцессора:

#include <stdio.h>

Функция fseek() — стандартная функция языка Си, устанавливает позицию в потоке **stream** для следующей операции чтения-записи на расстоянии **offset** байтов от позиции **ptrname**, принимающей следующие значения:

- 0 — начало
- 1 — текущая позиция
- 2 — конец

Функция **fseek()** возвращает ненулевое значение для недопустимых установок, 0 в противном случае.

Пример 1

```
fseek(fp, 0L, 0);                    /*установка в начало*/
fseek(fp, 0L, 2);                    /*в конец файла*/
fseek(fp, -sizeof(struct emp), 1);   /*возврат на одну структуру*/
```

Для реализации произвольного доступа к файлу бывает полезна функция **ftell()**.

Объявление прототипа функции **ftell()**:

long ftell(FILE *stream)

Функция ftell() — стандартная функция языка Си, которая возвращает смещение текущей позиции внутреннего указателя относительно начала файла (например, 0, если он указывает на начало файла). Для компилятора языка Си в ОС UNIX это смещение измеряется в байтах, для других операционных систем это может быть не так.

Пример 2

```
/*Печатает хранящуюся информацию о запрошенном объекте*/
#include <stdio.h>
struct part_info {
                    char desc[20];
                    int qty;
                    int cost;
                    }part;
int main()
{
  FILE *rptr;
  long x;

  if ((rptr=fopen("p_data","r"))==NULL)
     {
     fprintf(stderr,"Файл данных не может быть открыт\n");
     exit(1);
     }
  printf("Введите порядковый номер записи:");
  scanf("%ld", &x);
  fseek(rptr, ((x-1)*sizeof(part)), 0);
  fread (&part, sizeof(part), 1, rptr);
  printf("%s\t%d\t%d\n", part.desc, part.qty, part.cost);
}
```

Произвольный доступ при обновлении файла

Файл может использоваться как для чтения так и для записи, если он открыт для "обновления" (**"r+", "w+", "a+"**). В этом случае используется лишь один буфер, который должен вытесняться на диск между выполнением операций чтения и записи для обеспечения требуемой взаимной последовательности считываемых и записываемых данных. Для этого необходимо или явно вызывать функцию **fflush()**, или использовать функцию **fseek()**, которая, кроме установки позиции в файле для следующего чтения или записи, вызывает функцию **fflush()** для вытеснения буфера.

Пример 3

```
#include <stdio.h>
...
if ((rptr=fopen("p_data","r+")) == (FILE *)NULL)
  {
   fprintf(stderr,"Файл данных не может быть открыт\n");
   exit(1);
  }
/*изменяет все поля cost в учетном файле, составленном из структур*/
while (fread(&part, sizeof(part), 1, fp) != 0)
   {
    part.cost *= 1.07;
    fseek(fp, -sizeof(part), 1);
   /*на одну запись назад*/
    fwrite(&part, sizeof(part), 1, fp);
    fseek(fp, 0L, 1);
   /* вытесняет буфер перед следующим чтением */
  }
. . .
```

Функции обработки ошибок

Объявления прототипов функций обработки ошибок находятся в файле заголовков **stdio.h**, поэтому программы, использующие эти функции, должны включать следующую директиву препроцессора:

#include <stdio.h>

Каждая структура FILE содержит информацию о том, был ли достигнут конец соответствующего файла и о том, произошла ли ошибка ввода-вывода (например, попытка писать в файл, который был открыт только для чтения).

Функция feof() - стандартная функция ввода-вывода, которая проверяет достижение конца файла.

Объявление прототипа функции **feof()**:

int feof(FILE *stream)

Функция **feof()** возвращает отличное от нуля значение, если в процессе чтения или записи был достигнут конец файла, иначе она возвращает 0.

Пример 1

```
while (1)
   {
   n=fread(buf, 1, BUFSIZ, rp);
   fwrite(buf, 1, n, wp);
   if (feof(rp))
     break;}
```

Попытка записи в файл может быть безуспешной, если размер файла достиг максимума или исчерпалось свободное дисковое пространство.

Функция ferror() - стандартная функция ввода-вывода, которая выполняет проверку полей структуры **FILE**.

Функция clearerr() - стандартная функция ввода-вывода, которая выполняет очистку полей структуры **FILE**.

Объявления прототипов функциий **ferror()** и **clearerr()**:

```
int ferror(FILE *stream)
void clearerr(FILE *stream)
```

Функция **ferror()** возвращает отличное от нуля значение, если перед этим произошла ошибка ввода-вывода, иначе - 0.

Функция **clearerr()** сбрасывает признаки ошибки и конца файла в 0.

Пример 2

```
/*пытается MAX раз выполнить чтение с устройства*/
success = 0;
clearerr(rp);
for(i=1; i<MAX; i++)
  {
  fread(buf, BUFSIZ, 1, rp);
  if(!ferror(rp))
    {
    success = 1;
    break;
    }
  clearerr(rp);
  }
if (success)
  printf("Чтение выполнено\n");
else
  printf("После максимального числа попыток чтение не выполнено\n");
```