КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 12 — Форматный ввод/вывод

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2021

2021.11.10

Оглавление

Ввод/вывод <stdio.h>

Функции форматного вывода fprintf(), printf(), sprintf(), snprintf()

```
#include <stdio.h>

int fprintf(FILE * restrict stream,        // выводной поток
            const char * restrict  format, // строка формата
            ...);                          // список параметров вывода
int printf(const char * restrict format, ...);

int sprintf(char * restrict s,             // строка вместо потока
            const char * restrict format, ...);
int snprintf(char * restrict s,
             size_t cnt,         // не более cnt байт (включая конечный '\0')
             const char * restrict format, ...);
```

При успешном выполнении данные функции возвращают количество напечатанных символов без учета нулевого байта, используемого в конце выводимых строк, либо отрицательное значение при ошибке.

Функция **fprintf** записывает вывод в поток, на который указывает **stream**, под управлением строки, на которую указывает **format**.

stream — указатель на объект типа **FILE**, который идентифицирует выходной поток.

format указывает, как последующие аргументы (**...**) преобразуются для вывода.

Если для формата недостаточно аргументов, поведение не определено. Если формат исчерпан, пока остаются аргументы, избыточные аргументы вычисляются как обычно, но игнорируются.

Функция **fprintf** возвращает управление, когда встречается конец строки формата.

Строка **format** должна представлять собой многобайтовую последовательность символов, начинающуюся и заканчивающуюся в своем начальном состоянии сдвига. Т.е. строка синтаксически должна быть полной.

Строка **format** состоит из нуля или более директив следующих двух типов:

- обычные многобайтовые символы (за исключением **%**), которые копируются без изменений в выходной поток;

- *спецификации преобразования* (спецификатор формата), каждая из которых приводит к извлечению нуля или более последующих аргументов из списка переданных функции, преобразованию их, если это применимо, согласно соответствующему спецификатору преобразования, и затем записи результата в выходной поток.

Каждая спецификация преобразования начинается с символа %.

Формат спецификатора имеет следующий вид:

```
%[flags][width][.precision][length]specifier
```

После символа % появляется следующее:

1) **flags** — ноль или более *флагов* (в любом порядке), которые изменяют смысл спецификации преобразования.

2) **width** — необязательная минимальная *ширина поля*. Если преобразованное значение имеет меньше символов, чем ширина поля, оно дополняется до ширины поля пробелами либо слева (по умолчанию), либо справа, если задан флажок выравнивания по левому краю.

Ширина поля представляет собой неотрицательное десятичное целое число, либо звездочку * (ширина поля в этом случае задается аргументом). Если **width** задана числом, то 0 всегда принимается за флаг, а не за начало ширины поля.

```
%[flags][width][.precision][length]specifier
```

3) **precision** — необязательная точность представления, которая задает:
   а) минимальное количество цифр для преобразований **d**, **i**, **o**, **u**, **x** и **X**;
   б) количество цифр после знака десятичной точки для преобразований **a**, **A**, **e**, **E**, **f** и **F**;
   в) максимальное количество значащих цифр для преобразований **g** и **G**;
   г) максимальное число байтов, записываемых для **s** преобразований.

Точность имеет форму точки (**.**), за которой следует либо звездочка ***** (точность задается аргументом), либо необязательное десятичное целое число.

Если указана только точка, точность принимается равной нулю.

Если точность появляется с любым другим спецификатором преобразования, поведение не определено.

4) **length** — необязательный *модификатор длины*, который определяет размер аргумента.

5) **specifier** — символ *спецификатора преобразования*, который определяет интерпретацию аргумента и указывает тип преобразования, которое следует применить.

Ширина поля и точность

Ширина поля, или точность, или оба, могут быть обозначены звездочкой *****. В этом случае ширина поля и/или точность должны быть предоставлены аргументом с типом **int**.

Аргументы, указывающие ширину поля, и/или точность, должны появляться (именно в этом порядке) перед аргументом, подлежащим преобразованию.

Отрицательный аргумент ширины поля принимается как флаг, за которым следует положительная ширина поля.

Отрицательный аргумент точности принимается так, как если бы точность была опущена.

Спецификатор формата следует следующему прототипу:

```
%[flags][width][.precision][length]specifier
```

Спец-р.	Выводится в виде	Пример
d или **i**	Знаковое десятичное целое	392
u	Беззнаковое десятичное целое	7235
o	Беззнаковое восьмеричное целое	610
x	Беззнаковое шестнадцатиричное целое в нижнем регистре	7fa
X	Беззнаковое шестнадцатиричное целое в верхнем регистре	7FA
f	Десятичное с плавающей точкой в нижнем регистре	392.65
F	Десятичное с плавающей точкой в верхнем регистре	392.65
e	Десятичное с плавающей точкой в инженерном формате	3.9265e+2
E	Десятичное с плавающей точкой в инженерном формате	3.9265E+2
g	Десятич. с плав. точкой в наиболее коротком представлении **%e** или **%f**	392.65
G	Десятич. с плав. точкой в наиболее коротком представлении **%E** или **%F**	392.65
a	Шестнадцатиричное с плав. точкой в нижнем регистре	-0xc.90fep-2
A	Шестнадцатиричное с плав. точкой в верхнем регистре	-0XC.90FEP-2
c	Символ	a
s	Строка символов	пример
p	Адрес (значение указателя)	
n	Ничего не будет напечатано. Соответствующий аргумент должен быть указателем на **signed int**. Количество напечатанных символов хранится в указанном аргументом месте.	
%	**%**, за которым следует другой символ **%**, запишет в поток один **%**.	**%**

s — строка символов

Если модификатор длины **l** отсутствует, аргумент должен быть указателем на начальный элемент массива символьного типа[1].

Символы из массива записываются до завершающего нулевого символа, но не включая его.

Если указана точность, записывается не более указанного количества байт.

Ширина игнорируется.

Если точность не указана или превышает размер массива, массив должен содержать нулевой символ.

Если присутствует модификатор длины **l**, аргумент воспринимается как строка широких символов.

1 Особых условий для мультибайтных символов не предусмотрено

```c
#include <stdio.h>
#include <locale.h>
#include <errno.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

    char *ss = "qwertyuiop"; // 10 байтовых символов
    int rc = 0;
    setlocale(LC_ALL, "ru_RU.utf8");
    rc = printf("%s\n", ss);   printf("rc = %d\n", rc);
    rc = printf("%5s\n", ss);  printf("rc = %d\n", rc); // ширина игнорируется
    rc = printf("%.5s\n", ss); printf("rc = %d\n", rc); // точность определяет
                                                        // длину вывода
    rc = printf("%5ls\n", ss); printf("rc = %d\n", rc);
    if (rc < 0) {
        perror("%5ls "); exit(errno);
    }
    return 0;
}
```

```
qwertyuiop
rc = 11
qwertyuiop
rc = 11
qwert
rc = 6
rc = -1
%5ls : Неверный или неполный мультибайтный или широкий символ
```

p — адрес (значение указателя)

Аргумент должен быть указателем на **void**. Значение указателя преобразуется в последовательность печатных символов способом, определяемым реализацией.

n — ничего не будет напечатано

Аргумент должен быть указателем на целое число со знаком, в которое *записывается* количество символов, записанных в выходной поток к моменту обработки данного спецификатора данным вызовом **fprintf()**.

Никаких аргументов не преобразуется, но один из них «расходуется».

Если спецификация преобразования включает какие-либо флаги, ширину поля или точность, поведение не определено.

```
%[flags][width][.precision][length]specifier
```

Флаги

-	Выравнивание по левому краю в пределах заданной ширины поля. Правое выравнивание является значением по умолчанию.
+	Принудительно предшествует результату со знаком плюс или минус (**+** или **-**) даже для положительных чисел. По умолчанию только отрицательным числам предшествует знак **-**.
(space)	Если не будет записываться знак, перед значением будет вставлен пробел.
#	Если используется со спецификаторами **o**, **x** или **X**, значению предшествуют **0**, **0x** или **0X**, соответственно, для значений, отличных от нуля. Пример: **127 -> 0177, 0x7F, 0X7F; 255 -> 0377, 0xFF, 0XFF** Если используется со спецификаторами **a**, **A**, **e**, **E**, **f**, **F**, **g** и **G**, вынуждает записанный вывод всегда содержать десятичную точку, даже если за ней больше нет цифр. По умолчанию, если цифр за точкой нет, десятичная точка не записывается. Для преобразований g и G из результата не удаляются конечные нули. Для других преобразований поведение не определено.
0	Для преобразований **d**, **i**, **o**, **u**, **x**, **X**, **a**, **A**, **e**, **E**, **f**, **F**, **g** и **G** для дополнения ширины поля (если ширина задана спецификатором **width**) в начале вывода вместо пробела используются нули, которые будут следовать за знаком или основанием (кроме случаев преобразования бесконечности или NaN). Если появляются оба флажка 0 и -, флаг 0 игнорируется. Для преобразований **d**, **i**, **o**, **u**, **x** и **X**, если указана точность, флаг 0 игнорируется. Для других преобразований поведение не определено.

```
%[flags][width][.precision][length]specifier
```

width — ширина поля вывода (заполнения)

(число) Минимальное количество символов для печати. Если значение для печати короче этого числа, результат дополняется пробелами.
Если результат больше, значение не усекается.

* Ширина указывается не в строке формата, а в качестве дополнительного целочисленного аргумента, предшествующего аргументу, который должен быть отформатирован.

.precision — точность

.число Для целочисленных спецификаторов (**d**, **i**, **o**, **u**, **x**, **X**) точность определяет минимальное количество записываемых цифр. Если записываемое значение короче этого числа, результат дополняется начальными нулями. Если результат больше, значение не усекается. Точность 0 означает, что для значения 0 не написано ни одного символа.
Для спецификаторов **a**, **A**, **e**, **E**, **f** и **F** это количество цифр, которые будут напечатаны после десятичной точки (по умолчанию это 6).
Для спецификаторов **g** и **G** это максимальное количество значащих цифр для печати.
Для **s** это максимальное количество символов для печати.
По умолчанию все символы печатаются до тех пор, пока не встретится конечный нулевой символ.
Если точка указана без явного значения точности, предполагается 0.

.* Точность указывается не в строке формата, а в качестве дополнительного целочисленного аргумента, предшествующего аргументу, который должен быть отформатирован.

Пример форматирования по ширине и точности

```
// fmt.c
#include <stdio.h>

int    time_tag_width = 9;
double time_tag = 123.456789;
double tension = 1234.123456789;

int main() {

    printf("%12.6f: "
           "%+0*.2f\n",
           tension,
           time_tag_width, time_tag);
}
```

Компилируем и запускаем

```
$ gcc -std=c11 -W -Wall -pedantic -o fmt fmt.c
$ ./fmt
 1234.123457: +00123.46
$
```

Модификатор длины изменяет длину типа данных.

Ниже таблица, показывающая типы, используемые для интерпретации соответствующих аргументов с и без спецификатора длины.

Если используется другой тип, выполняется надлежащее продвижение или преобразование типа, если это разрешено.

	Спецификаторы						
длина	**d i**	**u o xX**	**fF eE gG aA**	**c**	**s**	**p**	**n**
нет	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	intmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

Следует обратить внимание, что спецификатор **c** принимает **int** (или **wint_t**) в качестве аргумента, но выполняет правильное преобразование в значение **char** (или **wchar_t**) перед форматированием его для вывода.

В зависимости от строки формата функция может ожидать последовательность дополнительных аргументов, каждый из которых содержит значение, которое будет использоваться для замены спецификатора формата в строке формата (или указатель на место хранения для **n**).

Таких аргументов должно быть как минимум столько же, сколько значений указано в спецификаторах формата. Дополнительные аргументы игнорируются функцией.

```
#include <stdio.h>

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

Функция **fprintf** записывает вывод в поток, на который указывает **stream**, под управлением строки, на которую указывает **format**.

Функция **printf** эквивалентна **fprintf** с аргументом **stdout** в качестве **stream**.

Функции **fprintf()** и **printf()** возвращают количество переданных символов или отрицательное значение, если произошла ошибка вывода или кодирования.

```
#include <stdio.h>

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

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

int snprintf(char *s,
             size_t n,
             const char *format, ...);
```

Функция **sprintf** эквивалентна **fprintf**, за исключением того, что выходные данные записываются в массив, заданный аргументом **s**, а не в поток **stream**. В конце записанных символов записывается нулевой символ, но он не учитывается при вычислении возвращаемого значения. Если происходит копирование между объектами, которые перекрываются, поведение не определено.

Функция **snprintf** эквивалентна **fprintf**, за исключением того, что выходные данные записываются в массив, заданный аргументом **s**, а не в поток. Если **n** равно нулю, ничего не записывается, также **s** может быть нулевым указателем. В противном случае выходные символы за пределами **n-1** отбрасываются и не записываются в массив.

В конце символов, фактически записанных в массив, записывается нулевой символ. Если копирование происходит между объектами, которые перекрываются, поведение не определено.

Функции **fprintf()** и **printf()** возвращают количество переданных символов или отрицательное значение, если произошла ошибка вывода или кодирования.

Функция **sprintf** возвращает количество символов, записанных в массив, не считая завершающий нулевой символ, или отрицательное значение, если произошла ошибка кодирования.

Функция **snprintf** возвращает количество символов, которые были бы записаны, если бы **n** было достаточно большим, не считая завершающий нулевой символ, или отрицательное значение, если произошла ошибка кодирования. Таким образом, вывод с нулевым символом в конце будет записан полностью тогда и только тогда, когда возвращаемое значение неотрицательно и меньше **n**.

Функции форматного ввода fscanf(), scanf(), sscanf()

```
#include <stdio.h>

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

int  scanf(const char * restrict format, ...);  // stdin

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

Функция **fscanf** считывает входные данные из потока, на который указывает **stream**, под управлением строки, на которую указывает **format**, который определяет допустимые входные последовательности и порядок их преобразования для присваивания, используя следующие за **format** аргументы в качестве указателей на объекты, которым подлежит получить преобразованный вход.

Функция **scanf** эквивалентна **fscanf** с аргументом **stdin**, в качестве входного потока, на который указывает **stream**.

Функция **sscanf** эквивалентна **fscanf**, за исключением того, что входные данные она получает из строки, указанной аргументом **s**, а не из потока.

Достижение конца строки эквивалентно обнаружению конца файла для функции **fscanf**.

Функции **`fscanf`**, **`scanf`** и **`sscanf`** возвращают **`EOF`**, если произошла ошибка ввода до завершения первого преобразования (в том случае если оно указано в формате).

В противном случае эти функции возвращают количество присвоенных элементов ввода, которое может быть меньше предусмотренного или даже равно нулю в случае досрочного неудачного сопоставления элементов ввода формату.

Если для формата недостаточно аргументов, поведение не определено.

Если формат исчерпан, а аргументы остаются, избыточные аргументы вычисляются (как всегда), но игнорируются.

```
#include <stdio.h>

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

Считывает данные из потока и сохраняет их в соответствии с форматом параметров в местах, указанных дополнительными аргументами.

Дополнительные аргументы должны указывать на уже выделенные объекты того типа, который указан соответствующим спецификатором формата в строке формата.

`stream` — указатель на объект **`FILE`**, который идентифицирует входной поток для чтения данных.

`format` — C-строка, содержащая последовательность символов, которая управляет обработкой символов, извлеченных из потока.

Формат состоит из нуля или нескольких директив:

1) один или несколько пробельных символов;

2) обычный многобайтовый символ, отличный от `%` (указывает на спецификатор формата) или пробельного символа;

3) спецификации преобразования.

1) Пробельные символы

Функция будет считывать и игнорировать любые пробельные символы (шпации), встречающиеся перед следующим непробельным символом (пробельные символы включают пробелы, символы новой строки и символы табуляции, как это трактуется функцией **`isspace( )`**).

Один пробел в строке формата соответствует любому количеству пробельных символов, извлеченных из потока, в том числе и не одного.

Директива, состоящая из символов пробела, исполняется путем чтения ввода до первого символа, не являющегося пробелом, который остается непрочитанным, или до тех пор, пока больше не будет прочитано больше символов.

Такая директива всегда завершается успешно.

2) Непробельный символ, за исключением спецификатора формата (`%`)

Любой символ, который не является ни пробельным символом (**`isspace( )`**), ни частью спецификатора формата (который начинается с символа **`%`**), заставляет функцию читать следующий символ из потока, сравнивать его с непробельным символом из строки формата, и, если они совпадают, символ из потока отбрасывается, и функция продолжает выполнение со следующего символа формата.

Если символы не совпадают, функция завершается ошибкой, возвращая и оставляя последующие символы потока непрочитанными. В случае конца файла, ошибки кодирования или ошибки директива завершается с ошибкой.

3) Спецификаторы формата

Последовательность, образованная начальным символом процента (%), представляет собой спецификатор формата, который используется для указания типа и формата данных, которые должны быть извлечены из потока и сохранены в местах, указанных дополнительными аргументами.

Каждая спецификация преобразования (спецификатор формата) представлена символом %.

После % может появляться следующее:

1) необязательный символ подавления присваивания '*'.

2) необязательное десятичное целое число больше нуля, указывающее максимальную ширину поля (в символах).

3) необязательный модификатор длины, который определяет размер принимающего объекта.

4) символ спецификатора преобразования, который указывает тип преобразования, которое будет применено.

Функция fscanf выполняет каждую директиву формата по очереди.

Когда все директивы выполнены или директива завершилась неуспешно, функция возвращает управление вызывающему.

Сбои характеризуются как сбои ввода в случае возникновения ошибки кодирования или недоступности вводимых символов (синтаксис), или сбои сопоставления из-за неправильного ввода (семантика).

Первый символ после элемента ввода, если он есть, остается непрочитанным.

Если длина элемента ввода равна нулю, выполнение директивы завершается неудачно. Эта ситуация является ошибкой сопоставления, но только в том случае, если ввод из потока не был прекращен по концу файла, ошибке кодирования или ошибке чтения.

В случае конца файла, ошибке кодирования или ошибке чтения это сбой ввода.

За исключением случая со спецификатором **`% (%%)`**, элемент ввода (или, в случае директивы **`%n`**, количество символов ввода) преобразуется в тип, соответствующий спецификатору преобразования.

Если не было указано подавление присваивания `*`, результат преобразования помещается в объект, на который указывает первый аргумент, следующий за аргументом **`format`**, который еще не получил результат преобразования.

Если этот объект не имеет подходящего типа или если результат преобразования не может быть представлен в объекте, поведение не определено.

Спецификатор формата для **fscanf** следует следующему прототипу:

```
%[*][width][length]specifier
```

Символ спецификатора в конце является наиболее значимым компонентом – он определяет, какие символы извлекаются, как их интерпретировать и тип соответствующего аргумента:

Спецификатор	Описание	Извлекаемые символы
i, u	целое	Считывает любое количество цифр, возможно с предшествующим знаком (+ или -). Десятичные цифры приняты по умолчанию (0-9). Префикс 0 означает восьмеричные цифры (0-7). Префикс 0x — шестнадцатеричные цифры (0-f).
d	десятичное целое	Любое количество десятичных цифр (0-9), необязательно с предшествующим знаком (+ или -).
o	восьмеричное целое	Считывает любое количество восьмеричных цифр (0-7), возможно с предшествующим знаком (+ или -).
x	шестнадцатиричное целое	Считывает любое количество шестнадцатиричных цифр (0-9, a-f, A-F), необязательно с предшествующим префиксом 0x или 0X, а также с необязательно предшествующим знаком (+ или -).

```
%[*][width][length]specifier
```

f, e, g, a	число с плавающей точкой	Считывает последовательность десятичных цифр, необязательно содержащая десятичную точку, которой необязательно предшествует знак (+ или -), за которым необязательно следует символ e или E и десятичное целое число (или некоторые другие последовательности, поддерживаемые функцией **strtod( )**). Реализации, соответствующие C99, также поддерживают шестнадцатеричный формат с плавающей запятой, когда ему предшествует **0x** или **0X**.
c	символ	Считывает следующий символ из потока. Если указана ширина, отличная от 1, функция считывает символы в точности согласно ширине и сохраняет их в последовательных местах массива, переданного в качестве аргумента. Нулевой символ в конец не добавляется.
s	строка символов	Считывает любое количество непробельных символов, останавливаясь на первом найденном пробельном символе. В конце сохраненной последовательности автоматически добавляется завершающий нулевой символ.
p	адрес, указатель	Последовательность символов, представляющая указатель. Конкретный используемый формат зависит от системы и реализации библиотеки, но он такой же, как формат, используемый для форматирования **%p** в **fprintf( )**.

[*символы*]	набор сканирования	Любое количество символов, указанное в скобках. Дефис (**-**), который не является первым символом, может вызывать непереносимое поведение в некоторых реализациях библиотеки.
[**^***символы*]		Любое количество символов, ни один из которых не указан в виде символов в скобках.
n	счетчик	Ничего не вводится. Количество символов, прочитанных из потока к данному моменту, сохраняется в указанном месте.
%	литеральный **%**	**%**, за которым следует другой **%**, соответствует одному **%**.

За исключением **n**, любой спецификатор должен использовать как минимум один символ. В противном случае совпадение не выполняется, и сканирование заканчивается в этом месте.

n

Ничего не вводится. Соответствующим аргументом должен быть указатель на целое число со знаком, в которое должно быть записано количество символов, прочитанных до этого момента из входного потока посредством данного вызова функции **fscanf**.

Выполнение директивы **%n** не увеличивает счетчик присвоений, возвращаемый при завершении выполнения функции **fscanf**. Ни один аргумент не преобразуется, но один используется. Если спецификация преобразования включает в себя символ подавления присваивания или ширину поля, поведение не определено.

Набор сканирования

[

Символ соответствует непустой последовательности символов из набора ожидаемых символов (*набор сканирования*) .

Спецификатор преобразования включает все последующие символы в строке формата, вплоть до соответствующей правой скобки (**]**).

Если модификатор длины **l** отсутствует, соответствующий аргумент должен быть указателем на начальный элемент массива символов, достаточно большой, чтобы принять последовательность и завершающий нулевой символ, который будет добавлен автоматически.

Если модификатор длины **l** присутствует, ввод должен представлять собой последовательность многобайтовых символов. Каждый многобайтовый символ преобразуется в широкий символ, как если бы он вызывался функцией **mbrtowc()**. Соответствующий аргумент должен быть указателем на начальный элемент массива **wchar_t**, достаточно большого, чтобы принять последовательность и завершающий нулевой широкий символ, который будет добавлен автоматически.

Символы в скобках (*список сканирования*), если символ после левой скобки не является галкой/крышей/гачиком/циркумфлексом (**^**), составляют набор сканирования.

Если символ после левой скобки является галкой/крышей/гачиком/циркумфлексом (**^**), в этом случае набор содержит все символы, которые не присутствуют в списке сканирования между **^** и правой скобкой.

Если спецификатор преобразования начинается с **[]** или **[^]**, считается что правый символ скобки **]** находится в списке сканирования, а завершает спецификацию следующий символ правой скобки, который считается совпадающей правой скобкой.

В остальных случаях первый же следующий символ правой скобки завершает спецификацию.

Если в списке сканирования присутствует символ **-** и он не является ни первым, ни вторым символом (в случае, когда первый символ является символом **^**), а также не последним, поведение определяется реализацией.

Никаких специальных условий для многобайтовых символов в правилах сопоставления, используемых в спецификаторах преобразования **c**, **s** и **[**, не предусмотрено — величина поля ввода определяется побайтно. Получающееся поле, тем не менее, представляет собой последовательность многобайтовых символов, которая начинается в начальном состоянии сдвига.