Содержание

1 Форматный ввод-вывод

Несколько библиотечных функций позволяют преобразовывать значения между внутренним представлением и текстовыми последовательностями, которые могут быть прочитаны и записаны. Аргументом `format` этих функций является *строка формата*, которая определяет требуемые преобразования. Функции делятся на две группы:

- Функции печати, объявленные в заголовочном файле `<stdio.h>`, преобразуют внутреннее представление в последовательности типа **`char`** и позволяют комбинировать последовательности для вывода. К этой группе относятся функции `fprintf`, `vfprintf`, `printf`, `vprintf`, `sprintf`, `vsprintf`, `snprintf`, `vsnprintf`.

- Функции сканирования, объявленные в `<stdio.h>`, преобразуют последовательности типа **`char`** и позволяют сканировать считываемые последовательности. К этой группе относятся функции `fscanf`, `scanf`, `sscanf`.

Строка формата состоит из нуля или более спецификаций преобразования, литерального текста и пробелов. Пробелы — это последовательность одного или более пробельных символов `c`, для которых вызов `isspace(c)` возвращает ненулевой результат.

Функции печати или сканирования просматривают форматную строку один раз от начала до конца для того, чтобы определить какие преобразования необходимо выполнить. Каждая из функций печати или сканирования принимает переменное число аргументов либо непосредственно, либо с помощью аргумента типа `va_list`. Некоторые спецификации преобразования в форматной строке используют следующий аргумент в списке аргументов. Функции печати или сканирования используют каждый аргумент не более одного раза. Аргументы, находящиеся в конце списка аргументов, могут остаться неиспользованными.

Стандарт C99 позволяет явно задавать, какой аргумент должен быть использован для преобразования `printf` во всех местах, требующих аргумент. Явное указание номера аргумента, содержащего преобразуемое значение, имеет вид `%`*m*`$`. Явное указание номера аргумента, содержащего ширину поля или точность, имеет вид `%`*m*`$`. В обоих случаях *m* — это десятичная запись целого числа, которая содержит хотя бы одну цифру, — порядковый номер аргумента, причём первый аргумент после форматной строки имеет номер 1. Таким образом, записи `printf("%*d", width, num);` и `printf("%2$*1$d", width, num);` эквиваленты. Новый стиль позволяет использовать один и тот же аргумент несколько раз. Стандарт C99 не позволяет смешивать оба стиля в одной спецификации форматного преобразования.

В дальнейшем:

- *Целочисленные преобразования* — спецификации преобразования, которые оканчиваются на `d`, `i`, `o`, `u`, `x` или `X`.

- *Вещественные преобразования* — спецификации преобразования, которые оканчиваются на `e`, `E`, `f`, `g` или `G`.

1.1 Строка формата функций печати

Для функций печати литеральный текст или пробельные символы в строке формата генерируют символы, совпадающие со строкой формата. Спецификация преобразования обычно генерирует символы, преобразовывая значение следующего аргумента в соответствующую текстовую последовательность. На рис. 1 показан синтаксис спецификаций преобразования функции `printf`.

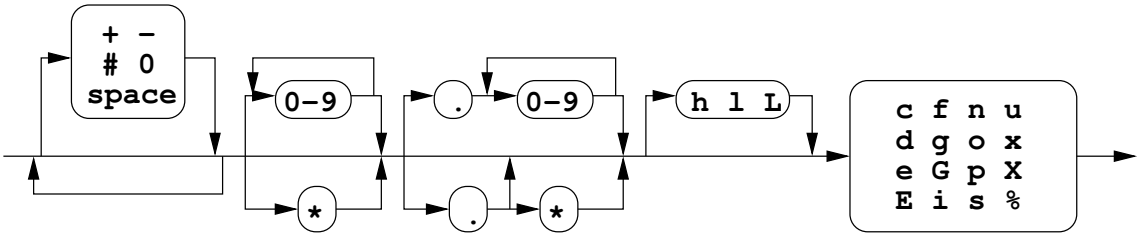

Рис. 1: Синтаксис спецификаций преобразования функции `printf` (**C90**)

Флаги. После знака процента `%` в форматной строке могут быть записаны ноль или более *флагов*:

- `-` для выравнивания по левому краю;
- `+` для печати знака «плюс» для положительных знаковых значений;
- *`space`* для печати пробела для знаковых значений, имеющих ни знака «плюс», ни знака «минус»;
- `#` для печати `0` («ноль») перед преобразованием `o` (восьмеричные числа), печати `0x` перед преобразованием `x`, печати `0X` перед преобразованием `X`, или для печати десятичной точки и цифр дробной части числа, в случаях, когда они бы не печатались в преобразованиях плавающей точки;
- `0` («ноль») для выравнивания преобразования ведущими нулями после любого знака или префикса в случае отсутствия флага `-` («минус») или спецификации точности.

Ширина поля. После всех флагов может быть записана *ширина поля*, которая задаёт минимальное количество символов, которые сгенерируются для данного преобразования. Если выравнивание явно не задано флагами, преобразование недостаточной длины по умолчанию заполняется слева пробелами. Если вместо десятичного числа записан символ `*`, функция печати использует следующий аргумент функции, который должен иметь тип **`int`**, как ширину поля. Если значение аргумента отрицательно, оно устанавливает флаг `-`, а его абсолютное значение задаёт ширину поля.

Точность. После спецификации ширины поля может быть записан знак `.` («точка»), за которым следует спецификация *точности* преобразования, которая обозначает: минимальное количество цифр для генерации по целочисленному преобразованию; количество

знаков после десятичной точки для преобразований `e`, `E`, `f`; максимальное количество значащих цифр для генерации по преобразованиям `g` или `G`; максимальное количество символов для печати из строки по преобразованию `s`.

Если вместо десятичного числа, задающего точность, записан знак `*`, функция печати использует следующий аргумент функции, который должен иметь тип **`int`**, как значение точности. Если значение аргумента отрицательно, используется точность по умолчанию. Если после знака **`.`** («точка») не записано ни десятичного числа, ни знака `*`, точность полагается равной 0.

Спецификатор преобразования. После точности следует *спецификатор преобразования*, состоящий из одного символа, перед которым может находиться односимвольный *квалификатор преобразования* (стандарт **C99** определяет несколько двухсимвольных квалификаторов). Каждая комбинация определяет тип, требуемый для следующего аргумента, и как аргумент меняется перед преобразованием его в текстовую последовательность. Целочисленные и вещественные преобразования кроме того определяют, какая система счисления используется для перевода в текстовое представление. Если спецификатор преобразования требует точности p, которая не задана в формате, выбирается точность по умолчанию, которая зависит от спецификатора преобразования. В таблицах 1, 2 перечислены комбинации и их свойства.

Спецификатор преобразования определяет всё поведение, не заданное в таблице. В описании, приведённом ниже, p — точность преобразования. Каждый спецификатор преобразования сопровождён примерами. Одно преобразование может генерировать до 509 символов (стандарт **C99** увеличивает предел до 4095 символов).

Спецификатор `%c`. Спецификатор используется для генерации одного символа из конвертированного значения.

Оператор	**Результат преобразования**
`printf("%c", 'a');`	`a`
`printf("<%3c|%-3c>", 'a', 'b');`	`<␣␣a|b␣␣>`

Спецификаторы `%d`, `%i`, `%o`, `%u`, `%x`, `%X`. Спецификаторы используются для генерации текстового представления целого числа возможно со знаком. `d` или `i` задаёт знаковое десятичное представление, `o` задаёт беззнаковое восьмеричное представление, `u` — беззнаковое десятичное представление, `x` — беззнаковое шестнадцатеричное представление, используя цифры `0`—`9` и `a`—`f`, `X` — беззнаковое шестнадцатеричное представление, используя цифры `0`—`9` и `A`—`F`. Преобразование генерирует минимум p цифр для представления преобразованного значения. Если p равно 0, преобразованное значение 0 не генерирует цифр.

Оператор	**Результат преобразования**
`printf("%d␣%o␣%x", 31, 31, 31);`	`31␣37␣1f`
`printf("%hu", 0xffff);`	`65535`
`printf("%#X␣%+d", 31, 31);`	`0X1F␣+31`

Спецификаторы `%e`, `%E`. Спецификаторы используются для генерации знакового дробного числа с показательной частью. Сгенерированный текст имеет вид $\pm d.d^pE\pm d^q$, где d — это десятичная цифра, E — это либо символ `e` для преобразования `e`, либо символ `E` для преобразования `E`. Сгенерированный текст содержит одну десятичную цифру, десятичную точку, если p не равен нулю, или задан флаг преобразования `#`, p цифр дробной части, q цифр ($q \geqslant 2$) показателя. Число перед преобразованием округляется до самого младшего показываемого знака. Значение 0 имеет экспоненту 0.

Спецификатор	Тип аргумента	Преобразованное значение	Основание	Точность	Примечания
hhd, hhi	**int** x	(**signed char**) x	10	1	C99
d, i	**int** x	(**int**) x	10	1	
hd, hi	**int** x	(**short**) x	10	1	
ld, li	**long** x	(**long**) x	10	1	
lld, lli	**long long** x	(**long long**) x	10	1	C99
Ld, Li	**long long** x	(**long long**) x	10	1	GNU
qd, qi	**long long** x	(**long long**) x	10	1	BSD
jd, ji	intmax_t x	(intmax_t) x	10	1	C99
td, ti	ptrdiff_t x	(ptrdiff_t) x	10	1	C99
hho	**unsigned int** x	(**unsigned char**) x	8	1	C99
o	**unsigned int** x	(**unsigned int**) x	8	1	
ho	**unsigned int** x	(**unsigned short**) x	8	1	
lo	**unsigned long** x	(**unsigned long**) x	8	1	
llo	**unsigned long long** x	(**unsigned long long**) x	8	1	C99
Lo	**unsigned long long** x	(**unsigned long long**) x	8	1	GNU
qo	**unsigned long long** x	(**unsigned long long**) x	8	1	BSD
jo	uintmax_t x	(uintmax_t) x	8	1	C99
zo	size_t x	(size_t) x	8	1	C99
hhu	**unsigned int** x	(**unsigned char**) x	10	1	C99
u	**unsigned int** x	(**unsigned int**) x	10	1	
hu	**unsigned int** x	(**unsigned short**) x	10	1	
lu	**unsigned long** x	(**unsigned long**) x	10	1	
llu	**unsigned long long** x	(**unsigned long long**) x	10	1	C99
Lu	**unsigned long long** x	(**unsigned long long**) x	10	1	GNU
qu	**unsigned long long** x	(**unsigned long long**) x	10	1	BSD
ju	uintmax_t x	(uintmax_t) x	10	1	C99
zu	size_t x	(size_t) x	10	1	C99
hhx, hhX	**unsigned int** x	(**unsigned char**) x	16	1	C99
x, X	**unsigned int** x	(**unsigned int**) x	16	1	
hx, hX	**unsigned int** x	(**unsigned short**) x	16	1	
lx, lX	**unsigned long** x	(**unsigned long**) x	16	1	
llx, llX	**unsigned long long** x	(**unsigned long long**) x	16	1	C99
Lx, LX	**unsigned long long** x	(**unsigned long long**) x	16	1	GNU
qx, qX	**unsigned long long** x	(**unsigned long long**) x	16	1	BSD
jx, jX	uintmax_t x	(uintmax_t) x	16	1	C99
zx, zX	size_t x	(size_t) x	16	1	C99

Таблица 1: Целочисленные спецификаторы преобразования `printf` и их свойства

Спецификатор	Тип аргумента	Преобразованное значение	Основание	Точность	Прим.
c	**int** x	(**unsigned char**) x			
e	**double** x	(**double**) x	10	6	
Le	**long double** x	(**long double**) x	10	6	
E	**double** x	(**double**) x	10	6	
LE	**long double** x	(**long double**) x	10	6	
f	**double** x	(**double**) x	10	6	
Lf	**long double** x	(**long double**) x	10	6	
g	**double** x	(**double**) x	10	6	
Lg	**long double** x	(**long double**) x	10	6	
G	**double** x	(**double**) x	10	6	
LG	**long double** x	(**long double**) x	10	6	
a	**double** x	(**double**) x	16	p	C99
La	**long double** x	(**long double**) x	16	p	C99
A	**double** x	(**double**) x	16	p	C99
LA	**long double** x	(**long double**) x	16	p	C99
n	**int** *x				
hn	**short** *x				
ln	**long** *x				
p	**void** *x	(**void** *) x			
s	**char** x[]	x[0]...		∞	
%	нет	'%'			

Таблица 2: Прочие спецификаторы преобразования printf и их свойства

Оператор	Результат преобразования
printf("%e", 31.4);	3.140000e+01
printf("%.2E", 31.4);	3.14e+01
printf("%e", 1e105);	1.000000e+105
printf("%e", 0.0);	0.000000e+00

Спецификатор %f. Спецификатор используется для генерации текстового представления знакового дробного числа без показательной части. Сгенерированный текст имеет вид $\pm d^q.d^p$, где d — это десятичная цифра. Сгенерированный текст содержит по крайней мере одну десятичную цифру, десятичную точку, если p не равен нулю, или задан флаг преобразования #, p цифр дробной части. Число перед преобразованием округляется до самого младшего показываемого знака.

Оператор	Результат преобразования
printf("%f", 31.4);	31.400000
printf("%0f %#f", 31.0, 31.0);	31 31.
printf("%3f", 1002.1);	1002.100000
printf("<%10f>", 2.4);	< 2.400000>

Спецификаторы %g, %G. Спецификаторы используются для генерации текстового представления знакового дробного числа с показательной частью или без неё. Для преобразования %g сгенерированный текст имеет ту же форму, как при преобразовании %e или %f. Для преобразования %G сгенерированный текст имеет ту же форму, как при преобразовании %E или %F. Точность p определяет количество генерируемых значащих цифр. Если p равно 0, оно заменяется на 1. Если преобразование %e дало бы показатель в диапазоне $[-4, p)$, применяется преобразование %f. Сгенерированный текст не имеет незначащих нулей в дробной

части числа и содержит десятичную точку только в случае ненулевого количества цифр дробной части, кроме случая, если задан флаг преобразования #.

Оператор	**Результат преобразования**
`printf("%.6g",␣31.4);`	`31.4`
`printf("%.1g",␣31.4);`	`3.14e+01`

Спецификаторы %a, %A. Спецификаторы используются для генерации текстового представления знакового вещественного числа с показательной частью. Мантисса и показатель записываются в шестнадцатеричной системе счисления. Сгенерированный текст имеет вид $\pm 0Xh.h^pP\pm h^q$, где X — это буква `x` для преобразования `%a` и буква `X` для преобразования `%A`; h — шестнадцатеричная цифра `0`—`9` или `a`—`f` для преобразования `%a` или шестнадцатеричная цифра `0`—`9`, `A`—`F` для преобразования `%A`; P — это буква `p` для преобразования `%a` и буква `P` для преобразования `%A`. Сгенерированный текст содержит одну шестнадцатеричную цифру, точку, затем p шестнадцатеричных цифр дробной части, причём точность по умолчанию достаточна для точного представления числа по основанию 2, если оно существует, либо достаточно велико, чтобы представить все биты числа типа **`double`** (**`long double`**, если указан квалификатор `L`). Шестнадцатеричная цифра перед точкой неопределена, если число ненормализовано, и не равна нулю, если число нормализовано.

Оператор	**Результат преобразования**
`printf("%a\n",␣0.0);`	`0x0p+0`
`printf("%a\n",␣16.125);`	`0x1.02p+4`
`printf("%A\n",␣1.45e+13);`	`0X1.A6016B2DP+43`
`printf("%La\n",␣-0.1L);`	`-0xc.cccccccccccccdp-7`
`printf("%a\n",␣1.0);`	`0x1p+0`

Спецификатор %n. Спецификатор используется, чтобы сохранить количество сгенерированных символов к тому месту, где в строке формата указан данный спецификатор. Значением следующего аргумента функции форматного преобразования должен быть адрес переменной соответствующего типа.

Оператор	**Сохранённое в x значение**	**Результат преобразования**
`printf("123%n4",␣&x);`	`3`	`1234`

Спецификатор %p. Спецификатор используется для генерации внешнего представления нетипизированного указателя (типа **`void*`**). Выполняемое преобразование зависит от реализации.

Оператор	**Результат преобразования (пример)**
`printf("%p\n",␣(void*)␣&x);`	`0xbffffa94`

Спецификатор %s. Спецификатор используется для генерации последовательности символов из значений, хранящихся в строке-аргументе функции. Преобразование генерирует не более чем p символов до символа конца строки `'\0'`, не включая его. Символ конца строки не учитывается при подсчёте количества генерируемых символов.

Оператор	**Результат преобразования**
`printf("%s",␣"hello");`	`hello`
`printf("%.2s",␣"hello");`	`he`
`printf("%.10s",␣"hello");`	`hello`
`printf("<%-7s>",␣"hello");`	`<hello␣␣>`
`printf("%2s",␣"hello");`	`hello`
`printf("<%7.2s>\n",␣"hello");`	`<␣␣␣␣␣he>`

Спецификатор %%. Спецификатор используется для генерации знака процента (`%`).

Оператор	Результат преобразования
`printf("%%");`	%

1.2 Функции печати

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const  char  *format, ...);

#include <stdarg.h>

int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vsprintf(char *str, const char *format, va_list ap);
int vsnprintf(char *str,size_t size, const char *format, va_list ap);
```

Функции форматного вывода форматируют выходной текст в соответствии с описанным выше форматом. Функции `printf` и `vprintf` выводят на стандартный поток вывода, `fprintf` и `vfprintf` выводят в заданный дескриптор потока, `sprintf`, `snprintf`, `vsprintf`, `vsnprintf` выводят в символьную строку `str`.

Функции `vprintf`, `vfprintf`, `vsprintf`, `vsnprintf` эквивалентны функциям `printf`, `fprintf`, `sprintf`, `snprintf` соответственно, за исключением того, что они вызываются с параметром `va_list` (список аргументов) вместо переменного количества аргументов. Данные функции не вызывают макрос `va_end`, значение `ap` после вызова неопределено. Вызывающая функция сама должна вызвать макрос `va_end(ap)`.

Функции возвращают количество напечатанных символов (не считая завершающий `'\0'`, используемый при выводе в строки). `snprintf` и `vsnprintf` не записывают более чем `size` байт (включая завершающий `'\0'`). Даже в случае, когда места в строке оказалось недостаточно, функции `snprintf` и `vsnprintf` возвращают количество символов (не считая `'\0'`, которое было бы записано, если бы строка имела достаточный размер.

Для функций `snprintf` и `vsnprintf` допускается указывать параметр `size`, равный 0, и параметр `str`, равный `NULL`. В этом случае функции возвращают длину строки, которая получилась бы при данном форматном выводе.

1.3 Строка формата функций сканирования

Для функций сканирования литеральный текст в форматной строке должен соответствовать следующим символам во входном тексте. Пробельные символы в форматной строке должны соответствовать последовательности максимальной длины из нуля или более пробельных символов во входном тексте. Все спецификаторы преобразования, кроме `n` (который не использует входной текст) определяют шаблон, которому должны соответствовать один или более следующих символов во входном тексте. Работа каждого спецификатора преобразования, кроме `c`, `n` и `[` начинается с пропуска пробельных символов во входном тексте.

Функция сканирования завершает работу при следующих условиях:

- она достигает конца форматной строки;

- она не может получить очередной символ входного текста (ошибка ввода);
- преобразование заканчивается неудачей (ошибка сопоставления).

Функция сканирования возвращает `EOF` (определённый в `<stdio.h>`), если ошибка ввода происходит до первого преобразования. В противном случае функция возвращает количество считанных значений. Если один или несколько символов образуют допустимый префикс, но преобразование завершается неуспешно, допустимый префикс прочитывается до завершения функции сканирования и не возвращается обратно в поток. Например,

`scanf("%i", &i)` прочитывает `0X` из поля `0XZ`
`scanf("%f", &f)` прочитывает `3.2E` из поля `3.2EZ`

Спецификация преобразования, как правило, конвертирует символы входного потока, удовлетворяющие формату спецификации, в соответствующее внутреннее представление. Соответствующим параметром функции сканирования при этом должен быть адрес объекта соответствующего типа. Преобразование конвертирует текстуальное представление значения во внутреннее и сохраняет его в указанном объекте. На рис. 2 показана синтаксическая диаграмма спецификаций преобразования функций сканирования.

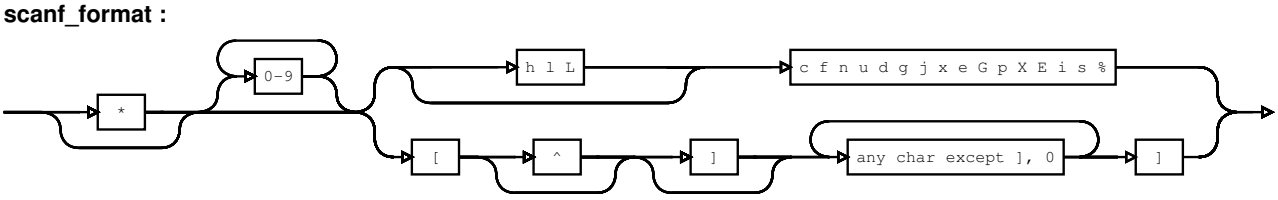

Рис. 2: Синтаксис спецификаций преобразования функции `scanf` (**C90**)

Флаги. После знака процента `%` может следовать знак `*`, означающий, что последующий спецификатор преобразования не должен сохранять преобразованное значение.

Ширина поля. После флагов может идти спецификация ширины поля, которая определяет максимальное количество входных символов, которые будут использоваться в преобразовании, не учитывая пробельных символов, которые могут быть пропущены согласно спецификации преобразования перед этим.

Спецификатор преобразования. За шириной поля должен следовать спецификатор преобразования: либо односимвольный код, либо множество сканирования, перед которым может быть указан одно- или двухсимвольный квалификатор преобразования. Каждый спецификатор преобразования определяет тип требуемого аргумента функции сканирования и то, как функция сканирования интерпретирует текстовую последовательность и преобразовывает её во внутренний формат. Целочисленные и вещественные преобразования также определяют, какая библиотечная функция будет вызвана для выполнения преобразования, и какое основание системы счисления подразумевается для текстового представления. Основание системы счисления — это аргумент `base` функций `strtol`, `strtoul`. В таблице 3 перечислены все определённые стандартом комбинации и их свойства.

В описании спецификатора преобразования или множества сканирования определено его поведение, не отражённое в таблице. В примерах, которые приводятся после описания каждого спецификатора преобразования, функция `sscanf` использует для форматного преобразования только подчёркнутые символы.

Спецификатор `%c`. Спецификатор преобразования `%c` используется для записи входных символов в массив элементов типа **`char`**. Если ширина поля ввода *w* не задана, используется

Спецификатор	Тип аргумента	Функция преобразования	Основание
d	**int** *x	strtol	10
hd	**short** *x	strtol	10
hhd	**signed char** *x	strtol	10
ld	**long** *x	strtol	10
lld	**long long** *x	strtoll	10
e, E, f, g, G	**float**	strtod	10
le,lE,lf,lg,lG	**double**	strtod	10
Le,LE,Lf,Lg,LG	**long double**	strtold	10
i	**int** *x	strtol	0
hi	**short** *x	strtol	0
hhi	**signed char** *x	strtol	0
li	**long** *x	strtol	0
lli	**long long** *x	strtoll	0
n	**int** *x		
hn	**short** *x		
hhn	**signed char** *x		
ln	**long** *x		
lln	**long long** *x		
o	**unsigned int** *x	strtoul	8
ho	**unsigned short** *x	strtoul	8
hho	**unsigned char** *x	strtoul	8
lo	**unsigned long** *x	strtoul	8
llo	**unsigned long long** *x	strtoull	8
u	**unsigned int** *x	strtoul	10
hu	**unsigned short** *x	strtoul	10
hhu	**unsigned char** *x	strtoul	10
lu	**unsigned long** *x	strtoul	10
llu	**unsigned long long** *x	strtoull	10
x, X	**unsigned int** *x	strtoul	16
hx, hX	**unsigned short** *x	strtoul	16
hhx, hhX	**unsigned char** *x	strtoul	16
lx, lX	**unsigned long** *x	strtoul	16
llx, llX	**unsigned long long** *x	strtoull	16
p	**void** **x		
c	**char** c[]		
s	**char** x[]		
[...]	**char** x[]		
%	нет		

Таблица 3: Спецификаторы преобразования функции `scanf`

значение по умолчанию $w = 1$. Это преобразование не пропускает ведущие пробельные символы. Шаблону преобразования соответствует любая последовательность из w символов. Символ-признак завершения строки `'\0'` после считанных символов не добавляется.

Вызов функции	Сохранённое значение
`sscanf("129E-2", "%c", &c)`	`'1'`
`sscanf("129E-2", "%2c", &c)`	`'1', '2'`

Спецификаторы `%d`, `%i`, `%o`, `%u`, `%x`, `%X`. Эти спецификаторы используются для преобразования входных символов в знаковое или беззнаковое целое число и сохранения его в объекте целого типа. Перед преобразованием пропускаются пробельные символы.

Вызов функции	Сохранённое значение
`sscanf("129E-2", "%o%d%x", &i, &j, &k)`	`10, 9, 14`

Спецификаторы `%e`, `%E`, `%f`, `%g`, `%G`. Эти спецификаторы используются для преобразования входных символов в знаковое вещественное число, состоящее из целой части, дробной части (возможно пустой) и необязательной экспоненциальной части. Результат помещается в объект одного из типов с плавающей точкой. Перед преобразованием пропускаются пробельные символы.

Вызов функции	Сохранённое значение
`sscanf("129E-2", "%e", &f)`	`1.29`

Спецификатор `%n`. Спецификатор используется для сохранения количества прочитанных на текущий момент символов в целочисленном объекте. Спецификатор не пропускает пробельные символы и не считывает входные символы. Он не учитывается в количестве успешно считанных полей, возвращаемом функциями сканирования.

Вызов функции	Сохранённое значение
`sscanf("129E-2", "12%n", &i)`	`2`

Спецификатор `%p`. Спецификатор используется для преобразования входных символов во внутреннее представление безтипового указателя (**`void*`**). Входные символы должны соответствовать формату спецификатора `%p` функций форматного вывода. Перед преобразованием пропускаются все ведущие пробельные символы.

Вызов функции	Сохранённое значение
`sscanf("129E-2", "%p", &p)`	`0x129e` (пример)

Спецификатор `%s`. Спецификатор используется для сохранения символов входного потока в массиве символов с добавлением за считанными символами признака завершения строки `'\0'`. Если ширина поля w не задана явно, по умолчанию полагается, что w велико. Перед чтением пропускаются пробельные символы. Считывается любая последовательность не более чем из w непробельных символов.

Вызов функции	Сохранённое значение
`sscanf("129E-2", "%s", &s[0])`	`"129E-2"`
`sscanf("129E-2", "%3s", &s[0])`	`"129"`

Спецификатор множества сканирования `%[...]`. Спецификатор используется для сохранения символов, удовлетворяющих шаблону, в массиве элементов типа **`char`** с добавлением после считанных символов признака завершения строки `'\0'`. Перед чтением символов не пропускаются пробельные символы. Если ширина поля ввода w не задана явно, по умолчанию предполагается, что w велико. Последовательность не более чем из w символов удовлетворяет шаблону, если выполняются следующие правила. После открывающей скобки `[` записывается последовательность из нуля или более символов множества сканирования, завершающаяся закрывающей скобкой `]`.

Если сразу же после `[` не записан символ «крышки» (`^`), тогда каждый вводимый символ должен соответствовать одному из символов множества сканирования. В противном слу-

чае каждый вводимый символ не должен соответствовать ни одному из символов множества сканирования, следующих после символа ^. Если символ] записан сразу же после [или ^, тогда этот символ] — один из символов множества сканирования, не символ, завершающий спецификатор множества сканирования. Символ «минус», записанный не первым и не последним символом во множестве определяет диапазон символов сканирования, например A-Z определяет множество символов, состоящее из заглавных латинских букв. Символ-терминатор строки '\0' не может быть использован в множестве сканирования.

Вызов функции	Сохранённое значение
`sscanf("129E-2", "%[12345]", &s[0])`	"12"
`sscanf("129E-2", "%[^EFG]", &s[0])`	"129"
`sscanf("129E-2", "%[0-9A-Fa-f]", &s[0])`	"129E"
`sscanf("129E-2", "%1[0-9A-Fa-f]", &s[0])`	"1"

Спецификатор %%. Спецификатор используется для сопоставления текущего вводимого символа с символом %. Спецификатор не сохраняет значения.

Вызов функции	Сохранённое значение
`sscanf("% 0xA", "%% %i", &i)`	10

1.4 Функции сканирования

```
#include <stdio.h>

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

Функции сканирования считывают входные символы согласно спецификации формата, описанной выше. Формат может содержать спецификации преобразования, результаты таких преобразований сохраняются по адресам, переданным в качестве параметров функций. Функция scanf считывает символы со стандартного потока ввода, fscanf считывает символы из заданного дескриптора потока, а функция sscanf считывает символы их строки, на которую указывает аргумент str.

Функции возвращают количество успешно выполненных спецификаций преобразования, предполагающих чтение входных символов. Это число может быть меньше, чем количество заданных спецификаций преобразования и даже 0. Ноль обозначает, что несмотря на то, что входные символы были доступны, форматных преобразований не было выполнено, чаще всего из-за недопустимого символа, встретившегося во вводе, например из-за алфавитного символа при преобразовании %d. Значение EOF возвращается, если ошибка ввода или конец файла произошли до первого форматного преобразования. Пропуск ведущих пробельных символов, выполняемый при определённых спецификациях преобразования, не считается началом форматного преобразования. Если ошибка ввода или конец файла произошли после того, как началось первое форматное преобразование, возвращается количество успешно выполненных до наступления ошибки или конца файла форматных преобразований. Для функции sscanf чтение символа-терминатора '\0' эквивалентно достижению конца файла.