

Основы программирования на С

Наумов Д.А., доц. каф. КТ

Операционные системы и системное программное обеспечение,
2020

Содержание лекции

- 1 Основные операторы языка
- 2 Объявления переменных
- 3 Условный оператор
- 4 Оператор выбора
- 5 Операторы циклов
- 6 Операторы перехода
- 7 Функции ввода/вывода

Операторы языка C

К основным операторам языка C относятся:

- Операторы-выражения;
- Описания переменных;
- Управляющие операторы.

Управляющие операторы задают порядок, в котором выполняются вычислительные операции программы. К управляющим операторам относятся:

- Условный оператор;
- Оператор выбора;
- Операторы цикла;
- Операторы безусловного перехода.

Операторы-выражения

Любое выражение, которое заканчивается точкой с запятой, является оператором.

Например:

```
func(); /* вызов функции */  
a = b + c; /* оператор присваивания */  
b + f(); /* правильный, но "странный" оператор */  
; /* пустой оператор */
```

В языке C точка с запятой является элементом оператора и его завершающей частью, а не разделителем операторов, как в языке Pascal.

Блоки

Блок - это последовательность операторов, заключенных в фигурные скобки и рассматривающихся как одна программная единица. Операторы, составляющие блок, логически связаны друг с другом. Иногда блок называют составным оператором. Блок всегда начинается открывающейся фигурной скобкой { и заканчивается закрывающейся }. Чаще всего блок используется как составная часть какого-либо оператора, выполняющего действие над группой операторов, например if или for. Однако блок можно поставить в любом месте, где может находиться оператор, как это показано в следующем примере:

```
#include <stdio.h>
int main(){
    int i;
    { /* блок операторов */
        i = 120;
        printf("\%d", i);
    }
    return 0;
}
```

Объявления переменных

Все переменные необходимо объявить до их использования. В объявлении указываются тип и список из одной или нескольких переменных этого типа:

```
int lower, upper, step;  
char c;
```

Переменные можно инициализировать прямо в объявлениях. Если после имени поставить знак равенства и выражение, то значение этого выражения будет присвоено переменной при ее создании:

```
int i = 0;  
int limit = MAXLINE + 1;
```

Объявление переменных может быть расположено в трех местах: внутри функции, в определении параметров функции и вне всех функций. Это - места объявлений соответственно локальных, формальных параметров функций и глобальных переменных.

Области видимости переменных

Чаще всего говорят о локальных и глобальных идентификаторах. Однако в языке C предусмотрено более тонкое подразделение этих двух широких категорий. Стандарт C определяет четыре типа областей видимости идентификаторов

Тип области видимости	Область видимости
Область действия - файл (имя, объявленное вне всех блоков, можно использовать в транслируемом файле, содержащем это объявление)	Начинается в начале файла и кончается в конце файла. Такую область видимости имеют только идентификаторы, объявленные вне функции. Эти идентификаторы видимы в любом месте файла. Переменные с этой областью видимости являются глобальными
Область действия - блок	Начинается открывающейся фигурной скобкой блока и кончается с его закрытием скобкой. Эту область видимости имеют также параметры функции. Переменные, имеющие такую область видимости, являются локальными в своем блоке
Область действия - прототип функции	Идентификаторы, объявленные в прототипе функции, видимы внутри прототипа
Область действия - функция	Начинается открывающейся фигурной скобкой функции и кончается с ее закрытием скобкой. Такую область видимости имеют

Квалификатор типа

В языке C определяются квалификаторы типа, указывающие на доступность и модифицируемость переменной. Стандарт C89 опеределеляет два квалификатора: `const` и `volatile`.

Квалификатор `const`

Переменная, к которой в объявлении применен квалификатор `const`, не может менять свое значение. Ее можно только инициализировать, то есть присвоить ей значение в начале выполнения программы. Компилятор может поместить переменную этого типа в постоянное запоминающее устройство. Например, в объявлении

```
const int a = 10;
```

создается переменная с именем `a`, причем ей присваивается начальное значение `10`, которое в дальнейшем в программе изменить никак нельзя.

Квалификатор `const` часто используется для того, чтобы предотвратить изменение функцией объекта, на который указывает аргумент функции. Без него при передаче в функцию указателя эта функция может изменить объект, на который он указывает. Однако если в объявлении параметра-указателя применен квалификатор `const`, функция не сможет изменить этот объект. Например:

```
size_t strlen(const char *str);
```

Квалификатор `volatile`

Квалификатор `volatile` указывает компилятору на то, что значение переменной может измениться независимо от программы, т.е. вследствие воздействия еще чего-либо, не являющегося оператором программы.

Например, адрес глобальной переменной можно передать в подпрограмму операционной системы, следящей за временем, и тогда эта переменная будет содержать системное время. В этом случае значение переменной будет изменяться без участия какого-либо оператора программы.

Знание таких подробностей важно потому, что большинство компиляторов C автоматически оптимизируют некоторые выражения, предполагая при этом неизменность переменной, если она не встречается в левой части оператора присваивания. В этом случае при очередной ссылке на переменную может использоваться ее предыдущее значение. Некоторые компиляторы изменяют порядок вычисления в выражениях, что может привести к ошибке, если в выражении присутствует переменная, вычисляемая вне программы. Квалификатор `volatile` предотвращает такие изменения программы.

Спецификаторы класса памяти

Стандарт C поддерживает четыре спецификатора класса памяти:

```
extern  
static  
register  
auto
```

эти спецификаторы сообщают компилятору, как он должен разместить соответствующие переменные в памяти. Общая форма объявления переменных при этом такова:

спецификатор_класса_памяти тип имя_переменной;

Спецификатор класса памяти в объявлении всегда должен стоять первым.

Спецификатор `extern`

Спецификатор `extern` указывает на то, что к объекту применяется внешнее связывание, именно поэтому он будет доступен во всей программе.

Далее нам понадобятся чрезвычайно важные понятия объявления и описания.

Объявление (декларация) объявляет имя и тип объекта. Описание (определение) выделяет для объекта участок памяти, где он будет находиться. Один и тот же объект может быть объявлен неоднократно в разных местах, но описан он может быть только один раз.

Спецификатор `extern` играет большую роль в программах, состоящих из многих файлов. В языке C программа может быть записана в нескольких файлах, которые компилируются отдельно, а затем компоуются в одно целое. В этом случае необходимо как-то сообщить всем файлам о глобальных переменных программы. Самый лучший (и наиболее переносимый) способ сделать это - определить (описать) все глобальные переменные в одном файле и объявить их спецификатором `extern` в остальных файлах, как показано в следующем примере.

Спецификатор extern

```
/* файл 1*/
int x, y;
char ch;
int main(void){
    /* ... */
}
void func1(void){
    x = 123;
}
```

```
/* файл 2 */
extern int x, y;
extern char ch;
void func22(void){
    x = y / 10;
}
void func23(void){
    y = 10;
}
```

Во втором файле спецификатор `extern` сообщает компилятору, что переменные `x`, `y`, `ch` определены в других файлах. Таким образом, компилятор узнает имена и типы переменных, размещенных в другом месте, и может отдельно компилировать второй файл, ничего не зная о первом. При компоновке этих двух модулей все ссылки на глобальные переменные будут разрешены. На практике программисты обычно включают объявления `extern` в заголовочные файлы, которые просто подключаются к каждому файлу исходного текста программы. Это более легкий путь, который к тому же приводит к меньшему количеству ошибок, чем повторение этих объявлений вручную в

Спецификатор `static`

Переменные, объявленные со спецификатором `static`, хранятся постоянно внутри своей функции или файла. В отличие от глобальных переменных они невидимы за пределами своей функции или файла, но они сохраняют свое значение между вызовами. Эта особенность делает их полезными в общих и библиотечных функциях, которые будут использоваться другими программистами. Спецификатор `static` воздействует на локальные и глобальные переменные по-разному. Статические локальные переменные - это локальные переменные, сохраняющие свое значение между вызовами функции. Спецификатор `static` в объявлении глобальной переменной заставляет компилятор создать глобальную переменную, видимую только в том файле, в котором она объявлена.

Спецификатор register

Первоначально спецификатор класса памяти `register` применялся только к переменным типа `int`, `char` и для указателей. Однако стандарт C расширил использование спецификатора `register`, теперь он может применяться к переменным любых типов.

В первых версиях компиляторов C спецификатор `register` сообщал компилятору, что переменная должна храниться в регистре процессора, а не в оперативной памяти, как все остальные переменные. Это приводит к тому, что операции с переменной `register` осуществляются намного быстрее, чем с обычными переменными, потому что такая переменная уже находится в процессоре и не нужно тратить время на выборку ее значения из оперативной памяти (и на запись в память).

В настоящее время определение спецификатора `register` существенно расширено. Стандарты C89 и C99 попросту декларируют "доступ к объекту так быстро, как только возможно". Практически при этом символьные и целые переменные по-прежнему размещаются в регистрах процессора. Конечно, большие объекты (например, массивы) не могут поместиться в регистры процессора, однако компилятор получает указание "позаботиться" о быстродействии операций с ними. в зависимости от конкретной реализации компилятора и операционной системы переменные `register`

Условный оператор if

Оператор if выполняет один из двух операторов в зависимости от значения некоторого условия.

```
if (выражение)
    оператор1
else
    оператор2
```

Часть, начинающаяся со слова else, необязательна.

В начале вычисляется выражение; если оно истинно (т.е. имеет ненулевое значение), то выполняется оператор1. Если оно ложно (т.е. имеет нулевое значение) и присутствует блок else, то выполняется оператор2.

```
if (a < b)
    z = a;
else
    z = b;
```

Точки с запятой - это часть операторов-выражений!

Оператор выбора switch

Оператор `switch` используется для выбора одного из нескольких вариантов действий в зависимости от того, с какой из набора целочисленных констант совпадает значение некоторого выражения.

```
switch (выражение){  
    case константное-выражение: операторы  
    case константное-выражение: операторы  
    default: операторы  
}
```

Блок `default` выполняется в том случае, если не найдено ни одного соответствия в блоках `case`.

Отличие от оператора `case` в языке Pascal состоит в том, что выражения `case` являются не отдельными ветвями алгоритма, а всего лишь метками, поэтому после перехода на одну из меток `case` будут выполнены все операторы после этой метки и до конца оператора `switch`, вне зависимости от наличия остальных меток `case`. Чтобы это предотвратить, в месте завершения ветви алгоритма нужно поставить оператор `break`.

Оператор цикла `while`

Цикл с предусловием `while` имеет формат:

```
while (выражение)  
    оператор
```

Выражение представляет собой условие продолжения цикла, которое вычисляется перед началом каждой итерации. Если выражение истинно, то выполняется очередная итерация цикла. Если выражение ложно, то цикл завершается.

Частный случай - бесконечный цикл:

```
while (1)  
    оператор
```

Оператор цикла do-while

Цикл do-while является циклом с постусловием, т.е. в нем условие проверяется после выполнения итерации цикла. Цикл do-while имеет формат:

```
do  
    оператор  
while (выражение);
```

Здесь сначала выполняется оператор, затем вычисляется выражение. Если оно истинно (не равно нулю), то снова выполняется оператор, и т.д. Как только выражение становится ложным, выполнение цикла прекращается.

За исключением способа проверки выражения, цикл do-while аналогичен оператору repeat-until в языке Pascal.

Оператор цикла for

Цикл for имеет следующий формат:

```
for (выраж1; выраж2; выраж3)  
    оператор
```

где выраж1 - инициализирующее выражение. Вычисляется один раз перед выполнением первой итерации цикла. Как правило содержит операции присваивания. Может содержать объявления переменных, которые будут существовать только в пределах цикла (в стандарте C99).

выраж2 - условие продолжения цикла. Вычисляется перед каждой итерацией цикла. Если выраж2 истинно, то выполняется очередная итерация, если ложно - то цикл завершается.

выраж3 - модифицирующее выражение. Вычисляется после каждой итерации цикла. Как правило содержит операции инкрементирования параметров цикла.

Любое из выражений может отсутствовать.

Оператор цикла for

Наиболее распространенный вариант цикла for (соответствующий циклу for в языке Pascal) имеет вид:

```
for (i = 0; i <= n; i++)  
    оператор
```

В языке C Допускаются и более сложные конструкции, например:

```
for (int i = 0, int j = 3; i + j < 100; i++, j+=2)  
    оператор
```

Если все три выражения в операторе for отсутствуют, то такой оператор представляет собой бесконечный цикл:

```
for (;;)   
    оператор
```

Метки. Оператор goto

В языке C имеется оператор `goto` - бездонный источник потенциальных неприятностей - и метки для перехода с его помощью. В техническом плане `goto` никогда не бывает необходим, и на практике почти всегда легче обходиться без него. тем не менее, в некоторых ситуациях оператор `goto` может упростить программу. Например:

```
for ( ... )  
    for ( ... ) {  
        if (disaster)  
            goto error;  
    }  
error:  
    решить проблему
```

Метка для перехода имеет ту же форму, что и имя переменной. После нее ставится двоеточие. Ее можно поставить перед любым оператором в той же функции, в которой находится соответствующий `goto`. Область действия метки - вся функция.

Операторы `break` и `continue`

Иногда бывает удобно выйти из цикла другим способом, отличным от проверки условия в его начале или в конце.

Оператор `break` вызывает принудительный выход из циклов `for`, `while`, `do-while` - аналогично выходу из оператора `switch`. Выход выполняется из ближайшего (самого внутреннего) цикла или оператора `switch`.

Оператор `continue` напоминает `break`, но используется реже: он передает управление на следующую итерацию ближайшего цикла `for`, `while`, `do-while`. В цикле `while` или `do-while` это означает немедленную проверку условия, тогда как в цикле `for` дополнительно выполняется инкрементирование.

Оператор возврата из функции `return`

Оператор `return` выполняет выход из функции и задает ее значение. Если функция возвращает значение, оператор `return` имеет формат:

```
return выражение;
```

В функциях типа `void` (не возвращающих значение) используется сокращенная форма:

```
return;
```


Функции ввода/вывода на консоль

В языке C нет специальных языковых конструкций для операций ввода/вывода данных. Операции ввода/вывода реализованы в виде библиотечных функций, для работы с которыми необходимо включить файл `<stdio.h>`.

Далее рассматриваются консольные функции ввода/вывода, т.е. те, которые выполняют ввод с клавиатуры и вывод на экран в текстовом режиме. В действительности же эти функции работают со стандартным потоком ввода и стандартным потоком вывода. Более того, стандартный ввод и стандартный вывод могут быть перенаправлены на другие устройства.

Рассматриваются:

- Чтение и запись символов;
- Чтение и запись строк;
- Форматный ввод/вывод на консоль (`printf()` и `scanf()`)

Чтение и запись символов

Простейшими функциями ввода/вывода являются функции `getchar()` и `putchar()`, предназначенные для ввода/вывода одиночных символов. Данные функции имеют следующие прототипы:

```
int getchar(void);  
int putchar(int);
```

Функция `getchar()` при каждом ее вызове возвращает следующий символ из потока или EOF в случае конца файла.

Функция `putchar()` помещает символ помещает символ в стандартный поток вывода, который по умолчанию соответствует экрану монитора. Функция `putchar()` возвращает выведенный символ или EOF в случае ошибки.

Пример: программа, переводящая поступающие данные в нижний регистр:

```
#include <stdio.h>  
int main(){  
    int c;  
    while ((c = getchar()) != EOF)  
        putchar(tolower(c));  
    return 0;
```

Чтение и запись строк

Функции `fgets()` и `fputs()` позволяют считывать и отображать строки символов.

Прототип функции `fgets()`:

```
char *fgets(char *str, int count, FILE *stream);
```

Функция `fgets()` считывает до `count` - 1 символов из входного потока `stream` и сохраняет их в `str`. В случае успеха возвращает указатель на строку, в случае ошибки - указатель на `NULL`.

Прототип функции `fputs()`:

```
int fputs(const char *str, FILE *stream );
```

Функция `fputs()` записывает строку `str` в поток вывода `stream`. В случае успеха функция возвращает неотрицательное значение, в случае ошибки - `EOF`.

Форматный ввод/вывод на консоль

Функции `printf()` и `scanf()` выполняют форматный ввод и вывод, то есть они могут читать и писать данные в разных форматах. Данные на консоль выводит `printf()`.

А ее “дополнение”, функция `scanf()`, считывает данные с клавиатуры.

Обе функции могут работать с любым встроенным типом данных, а также с символьными строками, которые завершаются символом конца строки (`'\0'`).

Функция printf()

Прототип функции printf():

```
int printf(const char *управляющая_строка, ...);
```

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

Управляющая_строка состоит из элементов двух видов. Первый из них - это символы, которые предстоит вывести на экран; второй - это спецификаторы преобразования, которые определяют способ вывода стоящих за ними аргументов. Каждый такой спецификатор начинается со знака процента, за которым следует код формата. Аргументов должно быть ровно столько, сколько и спецификаторов, причем спецификаторы преобразования и аргументы должны попарно соответствовать друг другу в направлении слева направо. Например, в результате такого вызова printf()

```
printf("Мне нравится язык %c %s 'С', "и к тому же очень сильно!");
```

Будет выведено

Мне нравится язык С и к тому же очень сильно

В этом примере первому спецификатору преобразования (%c) соответствует символ 'С', а второму (%s) - строка "и к тому же

Спецификаторы преобразования для функции printf()

Код	Формат
%a	Шестнадцатеричное число в виде 0xh.hhhhp+d (C99)
%A	Шестнадцатеричное число в виде 0Xh.hhhhP+d (C99)
%c	Символ
%d	Десятичное целое со знаком
%i	Десятичное целое со знаком
%e	Экспоненциальное представление ("e" в нижнем регистре)
%E	Экспоненциальное представление ("E" в верхнем регистре)
%f	Десятичное с плавающей точкой
%g	В зависимости от того, какой вывод будет короче, используется %e или %f
%G	В зависимости от того, какой вывод будет короче, используется %E или %F
%o	Восьмеричное без знака

Спецификаторы преобразования для функции printf()

Код	Формат
%s	Строка символов
%u	Десятичное целое без знака
%x	Шестнадцатеричное без знака (буквы в нижнем регистре)
%X	Шестнадцатеричное без знака (буквы в верхнем регистре)
%p	Указатель
%n	Аргумент, соответствующий этому спецификатору, должен быть указателем на целочисленную переменную. Спецификатор позволяет сохранить в эту переменную количество записанных символов (записанных до того места, в котором находится код %n)
%%	Знак %

Модификаторы формата функции printf()

Во многих спецификаторах преобразования можно указать модификаторы, которые слегка изменяют их значение. Например, можно указывать минимальную ширину поля, количество десятичных разрядов и выравнивание. Модификатор формата помещают между знаком процента и кодом формата. Здесь могут располагаться:

- Знак "минус задающий выравнивание выводимого аргумента по левому краю отведенного поля вывода.
- Число, задающее минимальную ширину поля. Преобразованный аргумент выводится в поле, ширина которого не меньше, чем заданная. Если необходимо, поле дополняется пробелами слева или справа до указанной длины.
- Точка, отделяющая ширину поля от точности представления.
- Число (точность представления), задающее максимальное количество символов при выводе строки, или количество цифр в вещественном числе после десятичной точки, или минимальное количество цифр для целого числа.
- Буква h, если целое число следует вывести как короткое (short), или буква l, если как длинное (long).

Пример. Модификаторы минимальное ширины поля

```
#include <stdio.h>
int main(){
    double item = 10.12304;
    printf("%f\\backslash n", item);
    printf("%10f\\backslash n", item);
    printf("%012f\\backslash n", item);
    return 0;
}
```

Вот что выводится при выполнении это программы:

```
10.123040
  10.123040
00010.123040
```

Пример. Выравнивание вывода

```
#include <stdio.h>
int main(){
    printf(".....\\backslash n");
    printf("по правому краю: \\%8d\\backslash n", 100);
    printf(" по левому краю: \\%-8d\\backslash n", 100);
    return 0;
}
```

Вот что выводится при выполнении это программы:

```
.....
по правому краю:      100
по левому краю: 100
```

Функция `scanf()`

Функция `scanf()` выполняет ввод с консоли. Она может читать данный всех встроенных типов и автоматически преобразовывать числа в соответствующий внутренний формат, `scanf()` во многом выглядит как обратная к `printf()`. Прототип функции `scanf()`:

```
int scanf(const char *управляющая_строка, ...);
```

Эта функция возвращает количество тех элементов данных, которым было успешно присвоено значение.

В случае ошибки `scanf()` возвращает EOF, управляющая_строка определяет преобразование считываемых значений при записи их в переменные, на которые указывают элементы списка аргументов. **Все аргументы функции `scanf()` должны быть указателями.**

Управляющая строка состоит из символов трех видов:

- спецификаторов преобразования,
- разделителей,
- символов, не являющихся разделителями.

Разделитель в управляющей строке дает `scanf()` указание пропустить в потоке ввода один или несколько начальных разделителей.

Разделителями являются пробелы, табуляции, вертикальные табуляции, подачи страниц и разделители строк.

Если в управляющей строке находится символ, не являющийся

Спецификаторы преобразования для функции scanf()

Код	Формат
%a	Значение с плавающей точкой (C99)
%c	Одиночный символ
%d	Десятичное целое число
%i	Читает целое число как в десятичном, так и восьмеричном или шестнадцатеричном формате
%e	Число с плавающей точкой
%f	Число с плавающей точкой
%g	Число с плавающей точкой
%o	Восьмеричное число

Спецификаторы преобразования для функции scanf()

Код	Формат
%s	Строка символов
%x	Шестнадцатеричное без знака (буквы в нижнем регистре)
%p	Указатель
%n	Принимает целое значение, равное количеству уже считанных символов
%i	Десятичное целое без знака
%[]	Набор сканируемых символов
%%	Знак %

Пример использования функции `scanf()`

Пусть необходимо вводить строки данных, в которых содержатся даты в следующем формате:

25 Dec 1988

Функция `scanf()` для этого случая будет выглядеть так:

```
int day, year;  
char month[20];  
scanf("%d %s %d", &day, &month, &year);
```

Считывать дату в формате мм/дд/гг из входного потока можно таким оператором:

```
int day, month, year;  
scanf("%d/%d/%d", &month, &day, &year);
```