НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика (основы программирования)»

Методические указания

Тематическое занятие 1

Данные, операции, программы, логические выражения, ветвления.

Содержание

Лексические соглашения

Лексика языка С

Язык С(Си), который рассматривается далее, определен стандартом ANSI 1989 года с учетом 1-й поправки, принятой в 1995 году.

Алфавитом называется совокупность символов, используемых в языке. *Лексемы* – неделимые последовательности знаков алфавита. Различают шесть видов лексем: идентификаторы, ключевые слова, константы, строковые литералы, знаки операций, разделители.

Ключевые (служебные, зарезервированные) слова языка С: `auto`, `break`, `case`, `char`, `const`, `continue`, `default`, `do`, `double`, `else`, `enum`, `extern`, `float`, `for`, `goto`, `if`, `int`, `long`, `register`, `return`, `short`, `signed`, `sizeof`, `static`, `struct`, `switch`, `typedef`, `union`, `unsigned`, `void`, `volatile`, `while`.

Разделители лексем (пробелы, табуляции, символы конца строки и прогона страницы, комментарии) игнорируются компилятором.

Комментарий – пояснительный текст, который не обрабатывается компилятором, ограничивается символами **`/*`** и **`*/`**.

Идентификаторы

Идентификаторы – имена (названия) переменных, функций, меток и других объектов. Различают стандартные и пользовательские идентификаторы. Идентификаторы составляются по следующим правилам:

- состоят из латинских букв, цифр и знака подчеркивания(_);
- начинаются с буквы или знака подчеркивания;
- не должны совпадать с зарезервированными словами;
- различимы прописные и строчные буквы (например, `MyVar`, `MYVAR`, `myvar` – имена различных переменных).

Примеры ***неправильных*** идентификаторов:

```
1program /* начинается цифрой */
block#1  /* содержит специальный символ */
My Prog  /* содержит пробел */
struct   /* зарезервированное слово */
```

Стандартные идентификаторы определены разработчиком языка. *Идентификаторы пользователя* определяет программист, причем нежелательно переопределять стандартные идентификаторы.

Программа на языке С

Пример простой программы

Рассмотрим программу, которая выводит на экран фразу «hello, world»:

```
#include <stdio.h>  /* подключение заголовочного файла */
main()  /* определение функции main без параметров */
{  /* начало блока функции main */
    printf("hello, world\n");  /* единственный оператор –
                                  вызов функции printf */
}  /* окончание блока функции main */
```

Функция `main` обязательно должна присутствовать в любой программе на языке С, с нее начинает выполняться программа. В данной программе в функцию `main` не передаются параметры, поэтому круглые скобки остаются пустыми. *Операторы* функции заключаются в фигурные скобки `{ }` – *блок* (или тело) функции.

Описание функции `printf` содержится в стандартном *заголовочном файле* ввода-вывода «`stdio.h`», который подключается к программе *директивой препроцессора* `#include`. В функцию `printf` передается единственный параметр (аргумент) – строковая константа (литерал) `"hello, world\n"`, где комбинация символов `\n` является условным обозначением для *символа конца строки*, который переводит курсор на левый край новой строки.

Вызов функции `printf()` становится оператором, когда после него ставится знак точка с запятой (`;`). Теперь программа может быть откомпилирована и выполнена.

Функция `main()`

Записанная без принятых сокращений функция `main` (без передачи в нее параметров), имеет следующий общий вид:

```
int main(void)
{
   /* объявления  и операторы */
   return 0;
}
```

Согласно стандарту языка С функция `main` всегда возвращает значение целочисленного типа `int`, а в случае успешного выполнения это значение должно быть равно нулю. Поэтому в конце тела функции `main` должен быть

оператор «`return 0;`», который определяет значение, возвращаемое в точку вызова.

Указанное в качестве параметра функции `main` ключевое слово `void` обозначает ***отсутствие значения***, т.е. в функцию не передаются параметры.

Типы данных, константы и переменные

Типы данных

Тип данных определяет множество значений, которые может принимать переменная, и совокупность операций, допустимых над этими значениями. Т.е. тип определяет форму внутреннего (машинного) представления данных и размер отводимой для них памяти.

Каждый тип данных имеет ограниченный диапазон значений. Выход за пределы этого диапазона вызывает *переполнение.*

В языке С имеется всего несколько следующих *базовых типов* данных:

`char` – *символьный* тип – один байт, содержащий один символ;
`int` – *целочисленный* тип (integer);
`float` – *вещественный* тип с плавающей точкой;
`double` – *вещественный* тип двойной точности с плавающей точкой.

Кроме того существуют модификаторы, которые могут применяться к базовым типам:

`short` – к целым числам;
`long` – к целым числам и типу `double`;
`signed` – к типу `char` и любому целочисленному;
`unsigned` – к типу `char` и любому целочисленному.

Причем вместо `short int` и `long int` можно указывать просто `short` и `long`, соответственно.

Размер памяти, которую занимает каждый из типов, выбирает компилятор в соответствии с характеристиками аппаратной части и набором ограничений, например, тип `int` должен иметь длину не менее 2 байт.

Каждый элемент данных является константой или переменной.

Константы

Константа (constant) – элемент данных, значение которого не изменяется в процессе выполнения программы. Значения констант задаются в тексте программы.

Целочисленная константа наподобие `1234` по умолчанию имеет тип `int` (если целое число слишком велико, то – тип `long`). Константу типа `long` записывают с суффиксом `L` (или `l`) на конце (например, `123456789L`); константу без знака – с суффиксом `U` (или `u`).

Вещественная константа (`123.4` или `1e-2` или `5.6e+2`) по умолчанию имеет тип `double`. Суффикс `F` (или `f`) обозначает константу типа `float`, а `L` (или `l`) – типа `long double`.

Символьная константа задается в виде символа в кавычках, например `'w'`. Ее значение – это целое число, равное коду данного символа.

Некоторые неотображаемые символы представляют с помощью *управляющих последовательностей,* например, `\n` – символ конца строки, `\t` – горизонтальная табуляция. Символьная константа `'\0'` представляет собой символ с нулевым кодом.

Строковая константа (*литерал*) – это последовательность символов, заключенных в двойные кавычки, например `"This is string."` или `"hello, world\n"`. А вот пустая строка – `""`.

Строковая константа является массивом символов. На длину строки не накладывается ограничений, в ее конец добавляется символ `'\0'`, который и служит признаком конца строки. Поэтому `'w'` – это числовой код буквы `w`, а `"w"` – это массив, содержащий один символ (букву `w`) и нуль `'\0'`.

Символические константы

Константам в языке С можно давать значащие имена с использованием механизма *макроопределения* (макроса). Следующая конструкция :

```
#define СимволическоеИмя ТекстДляПодстановки
```

определяет *символические имя* (символическую константу). Всякий раз, когда в программе встретится `СимволическоеИмя` (не в кавычках и не в составе другого имени), оно будет заменено на `ТекстДляПодстановки`, который может представлять собой последовательность любых символов.

Примеры задания символических констант с помощью этого механизма:

```
#define PI 3.1415  /* вещественная константа */
#define INCH 2.54  /* вещественная константа */
#define MAXN 1000  /* целочисленная константа */
```

Символические имена принято (хотя, не обязательно) составлять из прописных букв.

Объявление переменных

Переменная (variable) – область оперативной памяти, которой присвоено имя (идентификатор) для осуществления доступа к хранящимся в ней данным. Переменные могут менять свое значение в ходе выполнения программы.

В языке С все переменные необходимо *объявить до их использования.* В объявлении указывается тип и список из одной или нескольких переменных этого типа:

```
char     c;
int      a, b, sum;
float    radius, x, y;
short int          i;    /* тип с модификатором */
long double        x1, y1;
unsigned long int a1, b1;    /* тип с двумя модификаторами */
```

! Замечание: *Переменным следует давать такие* ***имена*** *(идентификаторы), которые* ***отражают назначение соответствующих переменных*** *(семантически* ***осмысленные****). Понятные идентификаторы являются своеобразной «****подсказкой****» и делают текст программы наглядным и выразительным.*

Инициализация переменных

В начале выполнения программы значения объявленных переменных не определены. Т.к. область памяти, выделенная для хранения переменных, может содержать произвольную последовательность нулей и единиц, то переменные могут принимать любые случайные значения из допустимого диапазона. Поэтому каждую переменная следует инициализировать до того, как она будет использована в качестве аргумента некоторого вычислимого выражения или оператора.

Инициализация переменной – присваивание ей некоторого определенного значения, которое является константой или константным выражением. При этом в языке С переменные можно инициализировать прямо в объявлениях:

```
int         k = 10;
float       r = 12.345;
char        letter = 'a';
int         a=2, b=3, c=5, d=8;
double      x = -1.23, y = 4.56 ;
long int    n = MAXN + 1;    /* константные выражения */
double      len = 0.3*INCH, angle = 2*PI/3;
```

Значение выражения, указанного после знака равенства, будет присвоено переменной при ее создании. При этом переменная не становится константой и может менять свои значения в ходе работы программы.

Операции и выражения, операторы

Арифметические операции

Язык С содержит богатый набор операций, которые обозначаются одним символом или набором символов. Среди них есть одно-, двух- и трехместные операции (по числу операндов). Каждая операция (operator) может иметь только определенные типы операндов.

Арифметические операции, применяемые к целочисленным и вещественным типам:

одноместные (унарные):	
+	*плюс*
-	*минус*
двухместные (бинарные):	
+	*сложение*
-	*вычитание*
*****	*умножение*
/	*деление* (для целочисленных типов – *частное от целочисленного деления*)
%	*остаток от целочисленного деления* (неприменима к вещественным типам)

Тип значения результата операции зависит от типа операндов. Например, при таком описании переменных:

```
int     a, b;
double  x, y;
```

результат операции деления будет разным:

`a/b` – частное от целочисленного деления `a` на `b` (целое число типа `int`),

`x/y` – обычное деление (вещественное число типа `double`).

В операциях в качестве операндов могут использоваться не только переменные, но и выражения.

Выражения

Выражения в языке С – это некоторая допустимая комбинация переменных, констант и операций (а также функций). Каждое выражение принимает какое-либо значение.

Порядок вычисления выражения определяется ***приоритетом*** и ***порядком ассоциирования (выполнения)*** операций, входящих в это выражение. Для арифметических операций:

Операции	***Ассоциирование***	***Приоритет***
( )	→ слева направо	высокий
+ - (унарные)	← справа налево	
*** / %**	→ слева направо	
+ - (бинарные)	→ слева направо	низкий

здесь все операции в одной строке таблицы имеют одинаковый приоритет. Заключение выражения в круглые скобки **( )** можно рассматривать как операцию, обладающую наивысшим приоритетом.

Например, при вычислении выражения:

```
a * (7 - a) / 3 + -b
  (3)   (1)   (4) (5)(2)
```

вначале (1) выполнится операция вычитания в скобках `(7-a)`, затем (2) будет вычислен результат унарной операции `-b`, потом (3) – умножение, далее (4) – деление, и последнее (5) – сложение. Если заранее было объявлено:

```
int a=5, b=2;
```

то результат вычисления выражения будет равен `1`. В данном случае вначале выполняется операция умножение (`*`), а затем – частное от целочисленного деления (`/`), хотя они имеют одинаковый приоритет. Такая последовательность соответствует порядку ассоциирования этих операций (слева → направо).

Вообще в языке С ***не регламентируется порядок вычисления операндов*** в операциях, что необходимо помнить, когда в качестве операнда выражения стоит вызов функции. (Примеры будут приведены в соответствующих разделах.)

В выражении языка С допускается использовать смешение переменных различных типов. Например, для описанных выше переменных допустимо:

```
5*a+x/(b-12.34)
```

При вычислении значения такого выражения действуют следующие правила преобразования типов.

Преобразования типов

Если операнды некоторой операции имеют различные типы, то они автоматически преобразуются к одному типу с использованием определенных в языке С правил. В целом эти правила заключаются в том, что прежде, чем

выполнится операция, более «узкий» базовый тип расширяется до более «широкого». Результат будет принадлежать более «широкому» типу.

Необходимо отметить, что при одновременном использовании знаковых `signed` и беззнаковых `unsigned` операндов существует вероятность получения некорректных результатов (в зависимости от системы, в которой компилируется и выполняется программа).

В любом выражении можно принудительно выполнить явное преобразование типов. Для этого используется одноместная операция *приведения типов*:

```
(ИмяТипа) выражение
```

При этом `выражение` как бы присваивается переменной заданного тапа, которая потом используется вместо всей конструкции. Например, для целых `a` и `b`:

```
(float) a
(double) 123+b*4
```

Приведение типов порождает новое значение заданного типа, никак не изменяя исходную переменную.

Приведение типов может использоваться в выражениях и имеет такой же приоритет, как у унарных `+` и `-`, ассоциирование тоже справа налево (←). Например, в выражении

```
(double) a/3
```

вначале будет выполнено приведение типов, а затем – деление. Т.е при целочисленном `a=2` результатом выражения будет значение `0.666667`, а не `0`.

При приведении типов в языке С также действуют определенные правила. Например, при преобразовании `float` в `int` отбрасывается дробная часть, а при преобразовании `double` в `float` значение числа либо усекается, либо округляется (в зависимости от конкретной реализации языка).

(Точные правила преобразований и приведений типов можно узнать в справочной литературе по языку С.)

Операция присваивания

Операция *присваивания* задает значения переменных в ходе выполнения программы. Она обозначается символом **=** и имеет синтаксис:

```
ИмяПеременной = Выражение
```

здесь `Выражение` – это константа, переменная, вычислимое выражение (или функция).

Порядок выполнения (ассоциирования) присваивания – справа налево (←): сначала вычисляется стоящее справа выражение, затем значение результата записывается в указанную слева. При этом могут происходить преобразования типов согласно описанным выше правилам.

В языке С ***операция присваивания может использоваться в выражениях***, и имеет самый низкий приоритет. Например, допустима запись

```
1 + 2 * (a = 3)
  (3)  (2)   (1)
```

здесь сначала `(1)` переменной `a` будет присвоено значение `3`, затем `(2)` будет выполнено умножение, а потом `(3)` – сложение.

Также допустимы выражения, наподобие следующего:

```
(a = 2) * a + a * (a = 3)
  (1)   (3) (5) (4)   (2)
```

Результатом вычисления выражения будет целое число `15`, а значение переменной `a` изменится дважды. Результат первой операции присваивания (`a=2`) будет использован только один раз, а второй (`a=3`) – трижды: `2*3+3*3=15`. Подобные выражения запутывают текст программы, и их следует избегать.

! Замечание: *Составлять выражения, зависящие от порядка вычисления операндов – это плохой стиль в любом языке программирования.*

Следующее выражение, наоборот, делает текст программы понятным и лаконичным

```
a = b = c = x*y
 (4) (3) (2)(1)
```

Такое многократное присваивание выполняется справа налево (в соответствии со своим порядком ассоциирования). Сначала `(1)` вычисляется `x*y`, затем `(2)` это значение присваивается `c`, потом `(3)` `b`, и лишь затем `(4)` `a`.

Операторы

В языке С некоторые выражения, наподобие `x=0` или `printf(…)`, становятся *операторами (statement)*, если после них поставить точку с запятой. Знак «точка с запятой» (`;`) не является разделителем операторов, а обозначает конец оператора. Таким образом, каждый отдельный оператор должен заканчиваться знаком «точка с запятой».

Очень часто присваивание выполняется как отдельный оператор – ***оператор присваивания***, для этого в конце выражения ставится точка с запятой:

```
ИмяПеременной = выражение;
```

Проследим за значениями переменных на примере следующей программы:

```
#include <stdio.h>
int main(void) {
   int    a, b=3, c; /* b=3, значения a и c - не определены */
   double x, y=3.6;  /* y=3.6, значение x - не определено */
   a=1+b*(c=2);           /* c=2, a=1+3*2=7 */
   x=a=2*a-(b=2*c+1);     /* b=5, a=9, x=9.0 */
   y=a/b+(double)a/b+a/y; /* y=9/5+9.0/5+9/3.6=1+1.8+2.5=5.3 */
   printf("a=%d, b=%d, c=%d; x=%f, y=%f\n", a, b, c, x, y);
                                          /* вывод на экран */
   return 0;
}
```

! Замечание: *Строки программы внутри разделов набраны с отступом (в несколько пробелов) для удобства восприятия (улучшения “читабельности”).*

Несколько операторов можно объединить в блок с помощью ***составного оператора (блока)***, заключив их между фигурными скобками `{ }`. За составным оператором знак «точка с запятой» не ставится.

Основные средства ввода-вывода

Стандартные функции ввода-вывода

Средства ввода-вывода не являются частью самого языка С, а содержатся в его стандартной библиотеке. Для форматированного вывода служит функция `printf`, для ввода – `scanf`. Эти функции описываются в заголовочном файле ввода-вывода «`stdio.h`», который подключается к программе *директивой препроцессора* `#include` :

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

Функции `printf` и `scanf` работают со стандартными ***потоками вывода*** и ***ввода***. Но мы пока будем считать, что это – экран монитора и клавиатура соответственно.

Функция форматированного вывода `printf()`

Стандартная функция `printf()` выводит на экран строку форматирования, указываемую в первом аргументе функции. Пример:

```
int x=45;
float y=1.234;
printf("Angle=%d deg. Length=%f m.\n", x, y);
```

Каждый знак `%` в этой строке обозначает **место**, куда будет подставлен следующий аргумент функции, а также **форму**, в которой этот аргумент будет выведен. В примере `%d` означает подстановку в данное место строки целого десятичного числа, а `%f` – вещественного числа с плавающей точкой.

В результате выполнения функции на экране появится:

```
Angle=45 deg. Length=1.234000 m.
```

Таким образом, строка форматирования состоит из элементов двух типов: 1) символы, которые непосредственно выводятся на экран; 2) *спецификаторы формата*, начинающиеся со знака `%`, которые определяют способ отображения аргументов функции `printf`.

Количество спецификаторов должно в точности совпадать с количеством аргументов функции `printf` и соответствовать с порядком их следования.

Спецификаторы и модификаторы

Основные спецификаторы функции `printf` указаны в таблице:

Специфи-катор	***Формат***
`%c`	символ
`%d`	десятичное целое со знаком
`%e`	экспоненциальное представление числа (в виде мантиссы и порядка), **`e`** – строчная
`%E`	экспоненциальное представление числа (в виде мантиссы и порядка), **`E`** – прописная
`%f`	десятичное вещественное число с плавающей точкой
`%o`	восьмеричное без знака
`%s`	символьная строка

Спецификатор	Формат
%u	десятичное целое без знака
%x	шестнадцатеричное без знака (строчные буквы)
%X	шестнадцатеричное без знака (прописные буквы)
%%	знак процента %

Формат вывода, использующего спецификаторы, можно изменять, помещая между знаком «%» и буквой спецификатора параметры, называемые *модификаторами*. Например, для рассмотренного случая:

Код с модификатором	Результат на экране	Описание вывода
%5d	`  45`	в 5 позиций *(впереди три пробела)*
%05d	`00045`	в 5 позиций *(впереди три нуля)*
%-5d	`45  `	по левому краю в 5 позиций *(позади три пробела)*
%10f	`  1.234000`	в 10 позиций *(впереди два пробела)*
%.4f	`1.2340`	с точностью 4 знака после запятой *(позади четыре пробела)*
%10.4f	`    1.2340`	в 10 позиций, с точностью 4 знака после запятой *(впереди четыре пробела)*
%-10.4f	`1.2340    `	по левому краю в 10 позиций, с точностью 4 знака после запятой *(позади четыре пробела)*

Имеются модификаторы для длинного и короткого типов данных:

Код с модификатором	Описание
%ld или %lf	для переменных с длинным типом (long int или double)
%hd	для переменных с коротким типом (short int)

Функция форматированного ввода scanf()

Стандартная функция scanf() считывает информацию с клавиатуры и сохраняет ее в переменных, перечисленных в списке аргументов. Первый аргумент – строка форматирования, которая задает правила чтения с клавиатуры.

Пример программы:

```
#include <stdio.h>
int main(void) {
   int x;
   float y;
   printf("Input angle(deg) and length(m): ");
   scanf("%d %f", &x, &y);
   return 0;
}
```

Здесь & – это операция получения адреса переменной, которая будет описана в дальнейшем в соответствующем разделе. Сейчас достаточно запомнить, что

символ `&` указывается перед **каждой** переменной в списке параметров функции `scanf`.

В приведенном примере вначале, используя функцию `printf`, на экран выводится строка запроса для пользователя. Затем функция `scanf` ожидает окончания ввода информации с клавиатуры: После чего значения, введенные пользователем, будут сохранены в переменных `x` и `y`.

Спецификаторы функций `scanf` и `printf` схожи, и практически совпадают.

(С подробностями работы стандартных функций ввода-вывода можно ознакомиться в справочной литературе по языку С.)

Примеры программ

Вычисление суммы двух целых чисел

```
#include <stdio.h>
int main(void) {
   int a, b;   /* объявление переменных a и b типа int */
   long int c; /* объявление переменной c типа long int */
   printf("Input a="); /* вывод строки запроса пользователя */
   scanf("%d", &a);    /* считывание введенного целого числа */
   printf("Input b=");
   scanf("%d", &b);
   c=a+b; /* вычисление суммы и присваивание результата переменной c */
   printf("Result c=%d\n", c);  /* вывод строки с результатом,
         значение переменной c помещается на место символов %d,
         а в конце добавляется символ перевода строки '\n' */
   return 0;
}
```

Форматированный вывод

```
#include <stdio.h>
int main(void) {
   int a = 12;  /* инициализация переменной a */
   double x;
   printf("Input real x: ");
   scanf("%lf", &x); /* %lf - считывание числа типа double */
   printf("a=%5d=%05d\n", a, a);
         /* форматированный вывод числа a на экран,
         %5d  - в 5 позиций,
         %05d - в 5 позиций с заполнением нулями */
   printf("x=%f=%.3f=%6.2f=%e\n", x, x, x, x);
         /* форматированный вывод числа x,
         %.3f  - с 3-мя цифрами после запятой,
         %6.2f - в 6 позиций с 2-мя цифрами после запятой,
         %e    - в форме записи с мантиссой */
   return 0;
}
```

Операции логических выражений

Операции отношения

Операции отношения выполняют сравнение двух операндов.

==	равно
!=	не равно (≠)
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

Логические операции

Логические операции и позволяют получать более сложные выражения.

!	отрицание «НЕ», NOT (*инверсия*)
&&	логическое «И», AND (*конъюнкция*)
||	логическое «ИЛИ», OR (*дизъюнкция*)

Результат выполнения логических операций задается *таблицей истинности*:

a	**b**	**! a**	**a && b**	**a || b**
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Приоритет операций

Таблица операций отношения и логических операций языка С в порядке приоритета:

Операции	***Ассоциирование***	***Приоритет***
( )	→ слева направо	высокий
!	← справа налево	
< <= > >=	→ слева направо	
== !=	→ слева направо	
&&	→ ***слева направо***	
||	→ ***слева направо***	низкий

Пример. Следующие записи выражения ***«*** `x` ***не лежит в диапазоне (–2; 2) »*** эквивалентны:

```
!(x>-2 && x<2)      и      x<=-2 || x>=2
```

Вычисление логического выражения

В языке С не существует отдельного логического типа данных. По определению числовое значение логического выражения или сравнения равно `1`, если выражение *истинно*, и `0` – если *ложно*.

Например, в результате выполнения фрагмента программы

```
int a;
a = 10;
printf("%d",a>5);
```

на экран будет выведено значение `1`.

Вычисление выражения, содержащего логические операции `&&` и `||`, выполняется *слева направо* и прекращается, как только установлено гарантированное значение результата (истина или ложь). Например, при `a=10` вычисление выражения с операцией `&&`

```
a>12 && (b=15)>a
```

окончится уже на первом операнде (`a>12` – ложь, а значит результат всего выражения – ложь, независимо от второго операнда), и присваивание (`b=15`) не будет выполнено. Вычисления выражений с логическими операциями происходят в этом порядке, даже не смотря на то, что скобки обладают более высоким приоритетом, чем `&&` и `||`.

Ветвления

Составной оператор

Составной оператор (блок операторов) – группа из произвольного числа операторов, которая ограничена *операторными скобками* – символами `{` и `}`. Составной оператор воспринимается как один оператор и используется там, где может стоять только один оператор, а требуется использовать несколько.

За составным оператором знак «точка с запятой» не ставится.

Условный оператор `if-else`

Алгоритмическая конструкция *ветвление* позволяет выбрать между несколькими вариантами действий в зависимости от заданного условия. Ветвление реализуется условным оператором и оператором выбора.

Условный оператор `if` имеет две синтаксических формы записи:

полная форма:

```
if (Выражение)
    ОператорИст
else
    ОператорЛож
```

сокращенная форма:

```
if (Выражение)
    ОператорИст
```

здесь: `ОператорИст` выполняется если `Выражение` истинно, `ОператорЛож` – если `Выражение` ложно. Ключевые слова `if`, `else` означают «**если**», «**иначе**» соответственно.

Выполнение условного оператора начинается с вычисления `Выражения`, которое считается истинным, если принимает ***ненулевое*** значение, и – ложным, если имеет нулевое значение. То есть для такой записи

```
if (a != 0) ...
```

предпочтительнее использовать более краткую форму

```
if (a) ...
```

поскольку в этом случае выполняется на одну операцию сравнения меньше, т.е. программа работает быстрее.

В такой конструкции `ОператорИст` (и `ОператорЛож`) может быть только одним оператором, блоком операторов или вообще отсутствовать (пустой оператор). Если требуется выполнить несколько операторов, то следует использовать составной оператор:

```
if (Выражение) {
    ОператорИст1
    ОператорИст2
    ...
    ОператорИстN
}
else {
    ОператорЛож1
    ОператорЛож2
    ...
    ОператорЛожM
}
```

Пример. Сравнение двух чисел:

```
#include <stdio.h>
int main(void) {
   int a, b;
   printf("Input a=");    scanf("%d", &a);
   printf("Input b=");    scanf("%d", &b);
   if (a>b)
      printf("a>b\n");
   else
      printf("a<=b\n");
   return 0;
}
```

! Замечание: *Для наглядности строки кода рекомендуется набирать с отступом. Причем, ключевое слово* `else` *стараются располагать строго под соответствующим ему* `if`.

Тернарная операция условия

Конструкции с условным оператором `if` можно записать другим способом, используя условное выражение с трехместной (тернарной) операцией «`?:`». Такое выражение имеет следующий вид:

```
ВыражУсл ? ВыражИст : ВыражЛож
```

Вначале вычисляется выражение `ВыражУсл`. Если оно истинно (т.е. не равно нулю), то вычисляется `ВыражИст`, значение которого становится значением всего условного выражения. В противном случае, вычисляется `ВыражЛож`, и его значение становится значением всего выражения. Всегда вычисляется только одно из `ВыражИст` и `ВыражЛож`.

Например, следующий код с условным оператором

```
if (a > b)
   c = a;
else
   c = b;
```

можно записать в виде выражения

```
c = (a > b) ? a : b ;    /* c = max(a,b) */
```

Скобки в первом операнде ставить необязательно, но рекомендуется, поскольку это улучшают восприятие текста программы.

Если выражения `ВыражИст` и `ВыражЛож` имеют различные типы, то тип результата определяется общими правилами преобразования типов, рассмотренными ранее. Например, если `x` имеет тип `float`, а `n` – тип `int`, то выражение

```
(n > 0) ? x : n
```

имеет тип `float` независимо от положительности значения `n`.

В выражениях приоритет тернарной операции условия самый низкий из всех рассмотренных, но выше, чем у операции присваивания. Порядок ассоциирования – справа налево (←).

Вложенность условного оператора

Один оператор `if` может входить в состав другого оператора `if`. *Вложенный* оператор может иметь вид:

```
if (Выраж1)
   if (Выраж2)
      ОператорИст2
   else
      ОператорЛож2
else
   ОператорЛож1
```

Ключевое слово `else` всегда ассоциируется с ближайшим предыдущим оператором `if` без `else`. Поэтому, если у вложенного `if` отсутствует раздел `else`, то нужно использовать блок:

```
if (Выраж1) {
   if (Выраж2)
      ОператорИст2
}
else
   ОператорЛож1
```

! Замечание: *Следует **избегать** конструкций с **вложенностью** условного оператора **более трёх уровней** из-за сложности их анализа при отладке программы. За исключением следующей конструкции* `else-if`.

Конструкция ***else-if***

Очень распространенный вариант вложенного условного оператора:

```
if (Выраж1)
   ОператорИст1
else if (Выраж2)
   ОператорИст2
else if (Выраж3)
   ОператорИст3
else
   ОператорЛож3
```

В программе сравнения двух чисел оператор `if` можно заменить:

```
if (a>b)
   printf("a>b\n");
else if (a<b)
   printf("a<b\n");
else
   printf("a=b\n");
```

Множественный выбор

Оператор ***switch***

Для выбора одного из нескольких вариантов действий используется оператор множественного выбора:

```
switch (Выражение) {
   case КонстантВыраж1: Операторы1
   case КонстантВыраж2: Операторы2
   ...
   case КонстантВыражN: ОператорыN
   default: ОператорыИначе
}
```

Если значение `Выражения` совпадает со значением одного из **целочисленных** константных выражений `КонстантВыраж1`, `КонстантВыраж2`, `...`, `КонстантВыражN`, то выполняются операторы, идущие после соответствующей метки `case`. Если не найдено ни одного соответствия, то выполняются операторы, идущие после метки `default`.

Блоки `case` – это, по сути, всего лишь метки, и после выполнения операторов в одном из них, продолжается выполнение операторов в следующем блоке `case`, пока не будет предпринят принудительный выход из `switch`. Такой немедленный выход может быть сделан оператором `break`. Использование оператора `switch` демонстрируется на следующем примере.

Пример множественного выбора

Программа, которая расставляет окончания слова «штука» в соответствии с правилами русского языка, в зависимости от числа, введенного пользователем:

```
#include <stdio.h>
int main(void) {
   int k;
   char flex;
   printf("Введите количество деталей:");
   scanf("%d", &k);
   switch (k) {
      case 1:
         flex = 'а';
         break;
      case 2:
      case 3:
      case 4:
         flex = 'и';
         break;
      default:
         flex = ' ';
         break;
   }
   printf("Количество деталей: %d штук%c\n", k, flex);
   return 0;
}
```

Если, например, убрать первый оператор `break`, то при `k=1` переменной `flex` вначале будет присвоено значение `'а'`, а затем – значение `'и'`. После этого будет выполнен выход из оператора `switch` по второму `break`, и выводимое сообщение будет неверно.

Такого ***сквозного*** выполнения оператора `switch` следует избегать, за исключением использования нескольких меток для одной и той же операции (как в приведенном примере, когда `k=2`, `3` или `4`).

Кроме того, следует ставить оператор `break` даже в конце последнего блока, хотя в этом нет необходимости. Это – хороший стиль программирования, который может подстраховать от лишних неприятностей при добавлении еще одного блока `case`.

Кстати, данная программа будет правильно расставлять окончания только для неотрицательных целых чисел `k`, не превосходящих значения `20`. Чтобы эта программа корректно работала для любых неотрицательных целых чисел, в операторе `switch` необходимо изменить `Выражение`, например, так:

```
   switch ( (k<=20) ? k : k%10 ) {
      ...  /* те же блоки case */
   }
```

! Замечание: *Вообще, **сквозной** метод программирования **не способствует** устойчивости программы к изменениям, поскольку при ее доработке могут возникать ошибки и побочные эффекты.*

Русский язык и локализация

В приведенном выше примере необходимо выводить русские буквы, для этого нужно переключиться на кодировку кириллицы.

Национальные стандарты могут быть подключены в ходе локализации с помощью заголовочного файла стандартной библиотеки `<locale.h>`. Используя описанную в этом заголовочном файле функцию `setlocale()` можно изменить текущую кодировку и правила форматирования чисел.

```
#include <stdio.h>
#include <locale.h>
int main(void) {
   setlocale(LC_ALL,"");
   printf("Вывод русских букв.\nРазделитель целой и дробной ");
   printf("части числа – запятая: %f\n",3.141592);
   return 0;
}
```

Упражнения

Упражнение 1.1

Составить программу, которая запрашивает у пользователя два целых числа и делит первое на второе.

Упражнение 1.2

Составить программу, которая запрашивает у пользователя два вещественных числа и выводит на экран их произведение с точностью до третьего знака.

Упражнение 1.3

Составить программу, которая выводит вещественные числа α=***123.45***, β=***9.876***, и γ=***45.6*** в столбик в следующим образом:

```
alpha = 123.45
 beta =   9.876
gamma =  45.6
```

При этом знаки «=» и «.» должны находиться точно друг под другом.

Упражнение 1.4

Составить программу, которая запрашивает у пользователя два целых числа и выводит на экран наибольшее из них. В случае если числа равны, программа не выводит ничего.

Упражнение 1.5

Составить программу, которая запрашивает у пользователя целое число и, если это число положительное и четное, выводит на экран соответствующее сообщение.

Упражнение 1.6

Составить программу, которая запрашивает у пользователя два целых числа. Если одно из чисел делит другое нацело (без остатка), то программа выводит на экран результат целочисленного деления, в противном случае программа выводит их сумму.