Условная инструкция

Все ранее рассматриваемые программы имели линейную структуру: все инструкции выполнялись последовательно одна за одной, каждая записанная инструкция обязательно выполняется.

Допустим мы хотим по данному числу x определить его абсолютную величину (модуль). Для этого необходимо нарушить линейную логику программы. Программа должна напечатать значение переменной x, если x>0 или же величину -x в противном случае. Линейная структура программы нарушается: в зависимости от справедливости условия x>0 должна быть выведена одна или другая величина. Соответствующий фрагмент программы на C++ имеет вид:

```
int x;
cin >> x;
if (x > 0)
{
    cout << x;
}
else
{
    cout << -x;
}
return 0;
```

В этой программе используется условная инструкция if (если). После слова if в обязательных круглых скобках указывается проверяемое условие (x > 0). После этого в фигурных скобках идет блок (последовательность) инструкций, который будет выполнен, если условие истинно, в нашем примере это вывод на экран величины x. Затем идет слово else (иначе) и после него блок инструкций, который будет выполнен, если проверяемое условие неверно, в данном случае будет выведено значение -x.

Итак, условная инструкция в C++ имеет следующий синтаксис:

```
if (Условие)
{
    Блок инструкций 1
}
else
{
    Блок инструкций 2
}
```

Блок инструкций 1 будет выполнен, если Условие истинно. Если Условие ложно, будет выполнен Блок инструкций 2.

В условной инструкции может отсутствовать слово else и последующий блок. Такая инструкция называется неполным ветвлением. Например, если дано число x и мы хотим заменить его на абсолютную величину x, то это можно сделать следующим образом:

```
if (x < 0)
{
```

```
        x = -x;
    }
```

В этом примере переменной x будет присвоено значение -x, но только в том случае, когда x<0.

Вложенные условные инструкции

Внутри условных инструкций можно использовать любые инструкции языка C++, в том числе и условную инструкцию. Получаем вложенное ветвление – после одной развилки в ходе исполнения программы появляется другая развилка. Покажем это на примере программы, которая по данным ненулевым числам x и y определяет, в какой из четвертей координатной плоскости находится точка (x,y):

```
    int x, y;
    cin >> x >> y;
    if (x > 0)
    {
        if (y > 0)   // x>0, y>0
        {
            cout << "Первая четверть" << endl;
        }
        else            // x>0, y<0
        {
            cout << "Четвертая четверть" << endl;
        }
    }
    else
    {
        if (y > 0)   // x<0, y>0
        {
            cout << "Вторая четверть" << endl;
        }
        else            // x<0, y<0
        {
            cout << "Третья четверть" << endl;
        }
    }
```

В этом примере мы использовали комментарии – текст, который компилятор игнорирует. Комментариями в C++ является последовательность символов // и весь текст после этого символа до конца строки. Обратите также внимание на отступы в начале строк, используемые для облегчения понимания текста.

Операторы сравнения

В качестве проверяемого условия должно использоваться выражение логического типа bool. Переменные логического типа принимают два значения: true (истина) и false (ложь). Также любое целочисленное выражение можно трактовать, как логическое выражение, при этом нулевое целое число означает ложь, а ненулевое — истину. Таким образом, если вместо условия написать false или 0, то оно будет всегда ложно, если же указать true, 1 или любое ненулевое число, то условие будет истинно.

Как правило, в качестве проверяемого условия используется результат вычисления одного из следующих операторов сравнения:

<
: Меньше — возвращает true, если первый операнд меньше второго.

>
: Больше — возвращает true, если первый операнд больше второго.

<=
: Меньше или равно.

>=
: Больше или равно.

==
: Равенство. Возвращает true, если два операнда равны.

!=
: Неравенство. Возвращает true, если два операнда неравны.

Например, условие (x * x < 1000) означает “значение x * x меньше 1000”, а условие (2 * x != y) означает “удвоенное значение переменной x не равно значению переменной y”.

Будьте аккуратны: оператор == (два знака равенства) — это проверка на равенство двух выражений, а оператор = (один знак равенства) — это присваивание одной переменной значения выражения и использование его в условии оператора ветвления в большинстве случаев является ошибкой.

Рассмотрим эту типичную ошибку на следующем примере:

```
int a, b;
cin >> a >> b;
if (a = b)
{
    cout << "Числа равны" << endl;
}
else
{
    cout << "Числа не равны" << endl;
}
```

Здесь по ошибке вместо операции сравнения == использована операция присваивания =. Поэтому при любых значениях a и b переменной a будет присвоено значение переменной b, при проверке истинности выражения a = b. Но оператор присваивания еще и возвращает значение, поэтому если значение b было ненулевым (а это интерпретируется, как истина), то программа выведет строку "Числа равны", а если нулевым — то строку "Числа не равны". При этом значение переменной a может быть вообще любым.

3.3 Логические операторы

Иногда нужно проверить одновременно не одно, а несколько условий. Например, проверить, является ли данное число четным можно при помощи условия (n % 2 == 0) (остаток от деления n на 2 равен 0), а если необходимо

проверить, что два данных целых числа n и m являются четными, необходимо проверить справедливость обоих условий: n % 2 == 0 и m % 2 == 0, для чего их необходимо объединить при помощи оператора && (логическое И):
n % 2 == 0 && m % 2 == 0.

В C++ существуют стандартные логические операторы: логическое И, логическое ИЛИ, логическое отрицание.

Логическое И является бинарным оператором (то есть оператором с двумя операндами: левым и правым) и имеет вид && (два знака амперсанда). Оператор && возвращает true тогда и только тогда, когда оба его операнда имеют значение true.

Логическое ИЛИ является бинарным оператором и возвращает true тогда и только тогда, когда хотя бы один операнд равен true. Оператор “логическое ИЛИ” имеет вид || (два знака вертикальной черты).

Логическое НЕ (отрицание) является унарным (то есть с одним операндом) оператором и имеет вид ! (восклицательный знак), за которым следует единственный операнд. Логическое НЕ возвращает true, если операнд равен false и наоборот.

Пример. Проверим, что хотя бы одно из чисел a или b оканчивается на 0:

```
if (a % 10 == 0 || b % 10 == 0)
```

Проверим, что число a — положительное, а b — неотрицательное:

```
if ( (a > 0) && !(b < 0) )
```

Или можно вместо !(b < 0) записать (b >= 0).

Другая типичная ошибка новичков — неверное использование двойных неравенств. Пусть нужно проверить, является ли введенное число ball корректной школьной оценкой, то есть числом от 1 до 5. Правильное решение такое:

```
if (ball >= 1 && ball <= 5)
```

А вот такая запись будет синтаксически верной, но работать будет неправильно:

```
if (1 <= ball <= 5)
```

Почему? Потому что в записи (1 <= ball <= 5) два оператора сравнения. Они выполняются слева направо и скобки между ними расставляются так:
((1 <= ball) <= 5). Сначала выполняется первое сравнение: (1 <= ball). Его результатом будет значение типа bool, то есть либо 0, либо 1. А на втором действии результат вычисления предыдущего выражения (то если либо 0, либо 1) сравнивается с числом 5. Поэтому итоговый результат всегда будет истинным, независимо от значения числа ball.