Инструкция ветвления

Инструкция ветвления обеспечивает выполнение одной или нескольких команд в зависимости от заданного логического выражения. Для ее записи используются два ключевых слова «`if`» и «`else`».

```
int a,b,c;
cin>>a>>b;
if (a<b)
  c=a;
else
  c=b;
cout << c << endl;
```

В круглых скобках записывается выражение, результатом которого является истина или ложь. Такие выражения называются логическими. Блок «`else`» не является обязательным и может отсутствовать.

Если в блоке условия требуется выполнить несколько инструкций их можно объединить при помощи фигурных скобок.

```
int a,b;
cin >> a >> b;
if (a>b) {
  int c=a;
  a=b;
  b=c;
}
cout << a <<' '<< b << endl;
```

Логические выражения

Для работы с логическими значениями язык C++ содержит специальный тип данных «`bool`» с двумя константами «`false`» и «`true`». Результатом всех операций сравнения «`<`», «`<=`», «`>`», «`>=`», «`==`», «`!=`» является значение, принадлежащее типу «`bool`».

В логических выражениях можно использовать операции: «`&&`» («И»), «`||`» («ИЛИ»), «`!`» («НЕ»). Например, программа проверки того, что из трех сторон можно составить прямоугольный треугольник може выглядеть так.

```
int a,b,c;
cin >> a >> b;
if (a*a==b*b+c*c || b*b==a*a+c*c || c*c==a*a+b*b)
  cout <<"rectangular triangle"<< endl;
```

Множественный выбор

В программировании часто возникает ситуация, когда требуется выбрать ровно один из нескольких вариантов дальнейших действий в зависимости он некоторого набора логических выражений. Для этого можно использовать следующую конструкцию.

```
if (...) {
  ...
} else if (...) {
  ...
} else if (...) {
  ...
```

```
} else {
   ...
}
```

Формально она является вложением нескольких инструкций ветвления, но логически ее можно воспринимать как одну инструкцию. Приведем пример фрагмента программы для вычисления знака числа `x`.

```
int x,sgnx;
cin>>x;
if (x<0)
  sgnx=-1;
else if (x>0)
  sgnx=1;
else
  sgnx=0;
cout << sgnx << endl;
```

Очень важно здесь не пропустить ключевое слово `else` при записи второго логического выражения, поскольку в этом случае программа будет синтаксически верной, но при этом будет выдавать неправильный ответ.

```
int x,sgnx;
cin>>x;
if (x<0)
  sgnx=-1;
if (x>0)
  sgnx=1;
else
  sgnx=0;
cout<<sgnx<<endl;
```

Здесь получилось две инструкции ветвления, которые не вложены, а выполняются последовательно. Поэтому, при отрицательных значениях `x` после выполнения первой инструкции, получим `sgnx=-1`, но после выполнения второй инструкции выполнится присваивание `sgnx=0`, и именно это значение и будет выведено в ответ.

Выбор оптимальной последовательности для вариантов исполнения программы может быть достаточно сложной задачей. Обычно хорошо работает такая эвристика: начинать проверку с тех условий, которые проверяются легче всего, так как после того как они будут проверены и отброшены, может упроститься и написание более сложных логических выражений.

Рассмотрим следующий пример. Дано целое число x. Если x является нечетным числом большим 1, то вывести $3x+1$; если x является четным числом большим 1, то вывести $x/2$. Для x меньших или равных 1 вывести само x.

Все три варианта работы являются взаимоисключающими и охватывают все множество целых чисел. Выберем наиболее простое условие из трех возможных. Очевидно, им будет `x<=1`, поэтому именно оно и будет записано первым. Теперь, в оставшихся случаях гарантировано $x > 1$, поэтому такую проверку можно далее не делать. Условия, проверяющие четность или нечетность числа не отличаются по сложности, поэтому вторым можно выбрать любое из них. Третий же вариант работы программы вообще не будет содержать проверок. Таким образом, программа примет следующий вид.

```
if (x<=1)
    cout<<x<<endl;
else if (x%2==0)
    cout<<x/2<<endl;
```

```
else
    cout<<3*x+1<<endl;
```

Структурное программирование

Структурное программирование — это парадигма, определяющая метод проектирования, написания и проверки программ, основанная на методе декомпозиции задачи на подзадачи. В основе структурного программирования лежит представление программы в виде блоков, имеющих иерархическую структуру. Это означает, что все блоки могут либо располагаться последовательно, либо быть либо вложенными друг в друга. Блоки в C++ выделяются при помощи фигурных скобок. Каждый блок выполняет некоторый логически связный набор действий, направленный на достижение определенного состояния, которое называют постусловием. Состояние до начала выполнения блока называют предусловием. Если два блока соединяются последовательно, то постусловие второго блока и предусловие первого блока совпадают.

В основе разработки программ в парадигме структурного программирования лежит принцип проектирования программ "сверху вниз". Это означает, что вся задача изначально разбивается на последовательность подзадач, после чего каждая из них проектируется и реализуется независимо. Если подзадача окажется слишком сложной, то она вновь может быть разделена на более мелкие подзадачи.

Рассмотрим пример структурной разработки программ на примере решения следующей задачи. На плоскости заданы три точки A, B, C с координатами (x_1, y_1), (x_2, y_2), (x_3, y_3) соответственно. Требуется определить, принадлежит ли точка C отрезку AB.

Программу можно разбить на две части: ввод исходных данных (**Подзадача 1.**) и решение и вывод ответа (**Подзадача 2.**). Мы не считаем здесь вывод ответа самостоятельной подзадачей, поскольку в данном случае он заключается в элементарном выводе слова "да"или "нет". Для проверки принадлежности точки C отрезку AB требуется проверить, что точка C принадлежит прямой AB, а также, что координаты C лежат между координат A и B. Таким образом, непосредственное решение разбивается еще на две подзадачи.

Подзадача 2.1. Принадлежность точки прямой может быть проверена по формуле векторного произведения для векторов AB и AC. Вектора AB и AC будут иметь координаты $(x_2 - x_1, y_2 - y_1)$ и $(x_3 - x_1, y_3 - y_1)$ соответственно. Векторное произведение будет равно $(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$ Точка принадлежит прямой тогда только тогда, когда векторное произведение равно нулю.

Подзадача 2.2. Для проверки того, что координаты C лежат между координатами A и B требуется проверить, что x_3 лежит между x_1 и x_2 (**Подзадача 2.2.1**), и, аналогично y_3 лежит между y_1 и y_2 (**Подзадача 2.2.2.**) Эти подзадачи являются одинаковыми, поэтому методология структурного программирования предполагает что они будут вынесены в отдельную подпрограмму, но здесь мы запишем их независимо.

Подзадача 2.2.1(2) Число не лежит между двумя другими, если оно меньше их обоих или больше их обоих. Возможны и другие решения, например, связанные с нахождением минимума или максимума.

Теперь приступаем к написанию программы.

```
#include <iostream>
using namespace std;
int main() {
  int x1,x2,x3,y1,y2,y3;
  // Подзадача 1. Ввод данных
  cin >> x1 >> y1;
  cin >> x2 >> y2;
  cin >> x3 >> y3;
```

```
  // Подзадача 2.1 Проверка векторного произведения
  if ((x2-x1)*(y3-y1)-(x3-x1)*(y2-y1)!=0)
    cout<<"no"<<endl;
  else { //Подзадача 2.2 Проверка С между A и B
    if ((x3<x1 && x3<x2 || x3>x1 && x3>x2)
      cout<<"no"<<endl;
    else if ((y3<y1 && y3<y2 || y3>y1 && y3>y2)
      cout<<"no"<<endl;
    else
      cout<<"yes"<<endl;
  }
  return 0;
}
```

Оформление кода

При оформлении кода программ следует уделять внимание удобству его чтения и проверки. Существует много стандартов оформления, каждая команда разработчиков обычно придерживается своего набора правил, поэтому здесь будут приведены наиболее распространенные и общие.

- Каждая строка должна содержать не более одной инструкции.
- Все инструкции последовательно выполняющиеся в одном блоке должны иметь одинаковый отступ от начала строки.
- Все инструкции входящие в состав вложенного блока должны иметь увеличенный отступ относительно того, в который они вложены. Это смещение должно иметь одинаковый размер для всей программы, обычно от 2 до 4 пробелов.
- Отступы от начала строки на протяжении всей программы должны формироваться либо только пробелами, либо только табуляцией.
- Имена идентификаторов на протяжении всей программы должны подчиняться одному определенному стилю. Заглавные и строчные буквы могут использоваться для определения вида идентификатора. Следует избегать использования символов близких по начертанию в тех случаях, когда это может вызвать путаницу, например `a1` и `al`.
- Имя переменной должно соответствовать ее содержанию.