В.Л. Тарасов

Лекции по программированию на C++

Лекция 3

Структура программ, управление

3.1. Состав программы

Программа на языке C++ состоит из функций и переменных. Среди функций любой программы должна быть функция `main()`. *Переменные* хранят данные, обрабатываемые программой.

Объявления переменных

Все переменные должны быть *объявлены* до их использования. Переменные можно объявлять внутри какой-либо функции или вне всех функций. Переменные, объявленные внутри функции, являются *локальными*, они существуют и доступны только внутри данной функции. Переменные, объявленные вне функций, являются *глобальными* или *внешними*, их можно использовать внутри любой функции.

Объявление задает тип и содержит список из одной или нескольких переменных. Например,

```
int lower, upper, step;  // Три переменные типа int
char c, line[1000], b;   // c, b – простые переменные типа char,
                         // line – массив из 1000 элементов типа char
```

При объявлении переменная может быть *инициализирована*, например,

```
char simb = 'G'; // В апострофы заключается одиночный символ
int i = 0;
double eps = 1.0e-5;
```

Объявления и определения

Объявление (declaration) сообщает компилятору о том, что в программе *будет* существовать какой-либо объект, например, переменная или функция. Инструкция:

```
extern int k;
```

является *только* объявлением. Теперь имя целой переменной k можно использовать в выражениях. Ключевое слово extern (внешний) указывает, что память под переменную k не выделяется при ее объявлении, а будет выделена где-то в другом месте.

Определение (definition) можно рассматривать как объявление, которое влечет за собой выделение памяти для переменной или функции. Объявленная выше переменная k должна быть определена где-то в программе инструкцией вида:

```
int k;
```

Все объявления переменных из предыдущего пункта являются также и определениями, так как компилятор зарезервирует под них память.

Один объект может объявляться несколько раз, но определение всегда единственное.

Когда не требуется конкретизация, вместо терминов *определение* и *объявление* можно использовать термин *описание*.

Инструкции и блоки

Программа на языке C++ состоит из отдельных предложений или *инструкций*. Каждая инструкция завершается *точкой с запятой* (;). Другое часто используемое название для инструкции – оператор, например, *оператор цикла*. Инструкция задает какое-то действие в программе, например, вызывает функцию, изменяет переменную.

Для объединения нескольких инструкций в одну составную инструкцию или *блок* используются фигурные скобки { и }. Блок с точки зрения синтаксиса воспринимается как одна инструкция.

В блоке можно объявлять переменные, которые существуют только внутри этого блока. Переменные внутри блоков «затеняют» переменные с такими же именами в охватывающих блоках. Это иллюстрируется в следующей программе.

Программа 3.1. Области видимости

```
int k = 2;                          // Глобальная переменная k
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{                                   // Блок с телом функции main
   cout << "k = " << k << endl;     // Использование глобальной переменной k
   int k = 0;                       // Создание локальной переменной k
   {                                // Внутренний блок
      int i;                        // i локализована во внутреннем блоке
```

```
      i = k;                          // Допустимо, k здесь доступна
      cout << "i = " << i << endl;    // Будет напечатано i = 0
      int k = 1;                      // Внутреннее k "затеняет" внешнее k
      i = k;                          // Используется внутреннее k
      cout << "i = " << i << endl;    // Будет напечатано i = 1
//    int k = 3;                  // Ошибка, в данной зоне видимости k есть
      cout << "::k = " << ::k << endl;        // Увидим  ::k = 0
//    cout << "::::k = " << ::::k << endl;    // Ошибка, выражение ::::k
                                              // недопустимо
   }                              // Конец блока - зоны видимости i
//   k = i + 1;                   // Ошибка, i недоступна вне своего блока
    ::k = 0;                      // Обращение к внешней переменной k
    cout << "::k = " << ::k << endl;
    system("pause");
    return 0;
}
```

Здесь два двоеточия (::) есть *оператор разрешения зоны видимости*, позволяющий обратиться к имени, которое недоступно в данном блоке, но существует во внешнем. С помощью этого оператора можно увидеть имя только в ближайшем охватывающем блоке.

Программа выводит:

```
k = 2
i = 0
i = 1
::k = 2
::k = 0
```

Переменные, используемые в качестве параметров цикла, можно определять непосредственно в заголовке цикла с необходимой инициализацией. Стандарт устанавливает областью действия таких переменных цикл. Например, в следующем фрагменте находится сумма первых 100 натуральных чисел.

```
  int s = 0;                          // Переменная для суммы
  for(int i = 1; i <= 100; i++)       // Зона действия i - цикл
     s += i;
  cout << i;                          // Ошибка, i недоступна после цикла
```

3.2. Константы

Константы – это объекты, которые нельзя изменить. Константы объявляются с добавлением ключевого слова const. При определении констант их следует инициализировать:

```
int main()
{
   const int Model = 90;      // model является константой
   const int X;               // Ошибка, нет инициализации
```

```
   Model = 200;              // Ошибка - попытка изменить константу
   extern const double F;    // Объявление константы
                             //  Определение будет в другом месте
   return 0;
}
const double F = 0.57;       // Определение константы
```

3.3. Условный оператор

Инструкции программы выполняются последовательно в том порядке, как они написаны. Естественный ход выполнения инструкций можно изменить с помощью управляющих операторов, одним из которых является условный оператор, обеспечивающий выполнение различных действий в зависимости от значения некоторого условия.

Условный оператор записывается в виде:

```
if(выр)
   инструкция1
else
   инструкция2
```

причем часть `else` может отсутствовать. Сначала вычисляется выражение `выр`. Если оно истинно (не нуль), то выполняется `инструкция1`, если `выр` ложно (нуль) и есть часть `else`, выполняется `инструкция2`.

Если имеется несколько вложенных условных операторов, то `else`-часть связывается с ближайшей `if`-частью. Например, во фрагменте:

```
if(n > 0)
   if(a > b)
      z = a;
   else
      z = b;
```

`else` относится к внутреннему `if`.
Для ясности можно использовать фигурные скобки:

```
if(n > 0){
   if(a > b)
      z = a;
   else
      z = b;
}
```

Для выбора одного из многих вариантов удобно записывать вложенные условные операторы в виде:

```
if(выр1)
   инструкция1
else if(выр2)
   инструкция2
else if(выр3)
```

```
      инструкция3
   else
      инструкция4
```

Как только встретится истинное выражение, выполняется соответствующая инструкция, и проверки завершаются. Последняя инструкция4 выполняется, если ни одно выражение не будет истинным.

Программа 3.2. Максимальное из двух чисел

В программе вводятся два целых числа, с помощью оператора if находится максимальное из них и выводится.

```
// Файл MaxOfTwo.cpp

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   int x, y, max;
   cout << "Введите первое число: ";   cin >> x;
   cout << "Введите второе число: ";   cin >> y;
   if(x > y)
      max = x;
   else
      max = y;
   cout << "Максимальное число max = " << max << endl;
   system("pause");              // Ожидаем нажатия любой клавиши
   return 0;
}
```

Далее приведен диалог с программой:

```
Введите первое число: 32
Введите второе число: 66
Максимальное число max = 66
```

3.4. Операторы цикла

Операторы цикла дают возможность многократно выполнять один и тот же участок программы, проводя одинаковые вычисления с различными данными. В языке C++ имеются три вида циклов: while, for и
do … while.

Цикл с предусловием while

Данный цикл записывается в виде:

```
while(выр)        // Заголовок цикла
   инструкция     // Тело цикла
```

Сначала вычисляется выражение выр. Если оно истинно (не нуль), выполняется инструкция и снова вычисляется выр. Повторения цикла завершаются, когда выр станет ложным (нулем).

Инструкция, выполняемая в цикле, называется *телом цикла*.

Так как условие повторения проверяется *до* выполнения тела цикла, данный цикл называют циклом *с предусловием*.

Тело цикла может не выполниться ни одного раза, если выр окажется ложным при первом его вычислении.

Программа 3.3. Суммирование цифр целого числа

Программа, вводит целое число и формирует другое число, равное сумме десятичных цифр первого.

Алгоритм решения задачи состоит в том, что находится последняя цифра числа, как остаток от деления числа на 10. Например, если исходное число n = 1234, то выражение n % 10 равно 4. Значение этого выражения добавляется к переменной m, начальное значение которой равно 0. Затем у числа n отбрасывается последняя цифра путем присваивания ему значения n / 10. Описанные действия повторяются, пока число n не станет равным нулю. Для организации повторений использован цикл while, а условием повторения является n > 0.

```
// Файл SumDigit.cpp

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   int n,                          // Заданное число
       sum = 0;                    // Сумма цифр
   cout << "Введите целое число: ";
   cin >> n;                       // Ввод числа
   if(n < 0){                      // Если число отрицательное
      cout << "Требуется положительное число \n";
      system("pause");;            // Ждем нажатия любой клавиши
      return 1;                    // Завершение программы
   }
   while(n > 0){                   // Пока в числе есть цифры, добавляем
      sum = sum + n % 10;          // к сумме последнюю цифру числа
```

```
      n = n / 10;                // и отбрасываем у числа последнюю цифру
   }
   cout << "Сумма цифр числа: " << sum << endl;
   system("pause");
   return 0;
}
```

После ввода числа n в условном операторе if проверяется, не меньше ли оно нуля. Если результат сравнения есть истина, то есть введено отрицательное число, выводится предупреждающее сообщение и функция main() завершается инструкцией return 1; при этом завершается и вся программа.

Возвращаемое функцией main() значение может быть обработано внешней программой, например, операционной системой. Принято возвращать 0 при нормальном завершении программы и значение, отличное от 0, при возникновении каких-то проблем.

Если введенное число положительно, программа выполняется дальше. Переменной sum присваивается начальное нулевое значение, в ней будет накапливаться сумма цифр. Последняя цифра, получаемая в цикле как остаток от деления числа на 10 оператором %, добавляется к сумме, затем последняя цифра числа отбрасывается путем его деления на 10 оператором /. Вот пример работы данной программы:

```
Введите целое число: 1234
Сумма цифр числа: 10
```

Цикл for

Конструкция цикла for имеет вид:

```
for(выр1; выр2; выр3)     // Заголовок цикла
   инструкция             // Тело цикла
```

Любое из трех выражений в скобках может отсутствовать, но точки с запятой опускать нельзя. Если отсутствует второе выражение выр2, то оно считается истинным. Цикл for эквивалентен конструкции:

```
выр1;
while(выр2){
   инструкция
   выр3;
}
```

Выражение выр1 вычисляется единственный раз перед началом работы цикла. Условием повторения цикла является истинность выр2. После выполнения инструкции в теле цикла вычисляется выр3.

Цикл for часто используют как цикл с параметром, обеспечивающий заданное число повторений, например, цикл:

```
for(i = 0; i < n; i++)
  …
```

повторяется ровно n раз. Начальное значение для i равно 0, после каждого выполнения тела цикла i увеличивается на 1 за счет выражения i++. Таким образом, i пробегает последовательность значений: 0, 1, 2,…, n-1, n. После завершения работы цикла его параметр i сохраняет последнее полученное значение i = n, которое можно использовать в дальнейших вычислениях.

Программа 3.4. Поиск максимума и минимума

Программа предлагает ввести заданное количество чисел, суммирует их, определяет максимальное и минимальное число и их номера в последовательности. В программе принято, что числа последовательности нумеруются начиная с нуля. Ввод и анализ данных выполняется в цикле for.

```
// Файл FindMaxMin.cpp

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   const int n = 5;          // Количество вводимых чисел
   int x;                    // Вводимое число
   int xmax, xmin;           // Максимальное и минимальное числа
   int nmax, nmin;           // Номера максимального и минимального чисел
   int sum = 0;                // Сумма
   cout << "Введите " << n << " чисел: "; // Приглашение к вводу
   cin >> x;                   // Ввод первого числа
   xmax = xmin = x;            // Предполагаем что первое число
   nmin = nmax = 0;            // и максимальное и минимальное
   sum = sum + x;              // Добавление числа к сумме
   for(int i = 1; i < n; i++){   // Цикл для ввода n-1 числа
      cin >> x;
      sum += x;                // Добавление числа к сумме
      if(x > xmax){            // Если встретилось большее число,
         xmax = x;             // запоминаем его и
         nmax = i;             // запоминаем его номер
      }
      else if(x < xmin){       // Если встретилось меньшее число,
         xmin = x;             // запоминаем это число
         nmin = i;             // и его номер
      }
   }                           // Конец цикла for
// Вывод результатов
```

```
    cout << "Максимальное число "  << xmax << ", его номер " << nmax;
    cout << "\nМинимальное число " << xmin << ", его номер " << nmin;
    cout << "\nСумма чисел sum = " << sum << endl;
    system("pause");
    return 0;
}
```

Для добавлении очередного числа к сумме использованы две конструкции: `sum = sum + x;` и `sum += x;`. Они эквивалентны, но вторая короче.

Пример работы программы:

```
Введите 5 чисел: 79 65 98 123 64
Максимальное число 123, его номер 3
Минимальное число 64, его номер 4
Сумма чисел sum = 429
```

Цикл do-while

Данный цикл записывается в виде:

```
do
    инструкция        // Тело цикла
while(выр)
```

Работа данного цикла состоит в следующем. Сначала выполняется `инструкция`, затем вычисляется выражение `выр` и проверяется его значение. Если `выр` истинно, снова выполняется `инструкция`. Когда `выр` становится ложным, цикл прекращает работу. Так как условие повторения цикла проверяется *после* выполнения тела цикла, он называется циклом *с постусловием*. Данный цикл целесообразно применять, когда тело цикла должно быть выполнено, по крайней мере, один раз.

Программа 3.5. Вычисление квадратного корня

Приближенное значение квадратного корня из некоторого положительного числа a можно найти с помощью последовательных приближений:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right), \quad n = 0, 1, 2, \ldots$$

Начальным приближением x_0 может быть любое положительное число, например, $x_0 = 1$ или $x_0 = a$. Итерационный процесс следует продолжать до тех пор пока два последовательных приближения не станут достаточно близки, что можно записать в виде:

$$|x_{n+1} - x_n| \le \varepsilon,$$

где ε – заданная малая величина – точность вычислений.

Далее приведен текст программы, реализующий процесс последовательных приближений для вычисления квадратного корня.

```
// Файл SqrtRoot.cpp

#include <iostream>
#include <locale>
#include <cstdlib>
#include <cmath>                        // Для fabs() (вычисляет модуль числа)
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   double a, xn, xn1, eps;
   do{                                  // Ввод и проверка исходных данных
      cout << "Введите число > 0: ";
      cin >> a;
      cout << "Введите точность > 0: ";
      cin >> eps;
   }while(a <= 0 || eps <= 0);          // Повторяем ввод при поступлении
                                        // отрицательного числа или точности
   xn1 = a;                             // Начальное приближение
   cout << xn1 << endl;
   do{
      xn = xn1;                     // Следующее приближение делаем предыдущим
      xn1 =  (xn + a / xn) / 2.0; // Новое приближение
      cout << xn1 << endl;
   }while(fabs(xn1 - xn) > eps);  // Пока не достигнута нужная точность,
   system("pause");               // Ждем нажатия любой клавиши
   return 0;
}
```

При вычислении $\sqrt{2}$ программа выдает следующее:

```
Введите число > 0: 2
Введите точность: 0.0001
2.000000
1.500000
1.416667
1.414216
1.414214
```

Видно, что уже пятое приближение имеет заданную точность.

Обсудим данную программу.

Включение заголовочного файла `math.h` необходимо, так как в программе имеется обращение к объявленной в этом файле функции

```
double fabs(double x);
```

вычисляющей абсолютное значение для числа с плавающей точкой.

Число a, два последовательных приближения к квадратному корню xn, xn1 и точность eps определены как числа с плавающей точкой двойной точности double.

Ввод исходных данных производится в цикле do … while. Этот цикл повторится, если будет истинным выражение a < 0 || eps <= 0. Оператор || является логическим оператором ИЛИ и все выражение будет истинным, если введено или отрицательное значение для a, или для точности eps. Синтаксис языка требует, чтобы между do и while была единственная инструкция. Здесь четыре инструкции объединены в одну составную инструкцию фигурными скобками { и }.

Вычисления начинаются с присваивания переменной xn1 значения a. Далее вычисления продолжаются в цикле do while.

Важным приемом программирования является присваивание:

```
xn = xn1;
```

которое позволяет обойтись только двумя переменными xn и xn1 при вычислении многих последовательных приближений.

Каждое из приближений печатается.

3.5. Переключатель

Для выбора одного из многих вариантов удобно использовать оператор switch, который имеет вид:

```
   switch(выр){
      case константа1  :
        инструкция1
      break;
      case константа2  :
         инструкция2
      break;
…
      case константаN  :
         инструкцияN
      break;
      default: инструкция
}
```

Ветвь default может отсутствовать.

Работа переключателя состоит в следующем. Вычисляется выражение выр, затем его значение сравнивается с константами. При совпадении выполняется соответствующая инструкция и работа завершается. Если выр не совпадает ни с одной константой, выполняется инструкция ветви default, если она есть.

Программа 3.6. День недели

Программа предлагает ввести номер дня недели и выводит его название.

```
// Файл DayWeek.cpp

// Программа выводит название дня недели по его номеру

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   int day;                    // Номер дня недели
   cout << "Введите номер дня недели от 1 до 7 \n";
   cin >> day;
   cout << "Это ";
   switch(day){
      case 1:  cout << "понедельник\n";   break;
      case 2:  cout << "вторник\n";       break;
      case 3:  cout << "среда\n";         break;
      case 4:  cout << "четверг\n";       break;
      case 5:  cout << "пятница\n";       break;
      case 6:  cout << "суббота\n";       break;
      case 7:  cout << "воскресенье\n";   break;
      default: cout << "неверный номер\n ";
   }
   system("pause");           // Ждем нажатия клавиши
   return 0;
}
```

Ниже приводятся результаты двух запусков программы.

```
Введите номер дня недели от 1 до 7
5
Это пятница

Введите номер дня недели от 1 до 7
8
Это неверный номер
```

Если нужно выполнить одно действие для нескольких констант, они последовательно перечисляются, как это демонстрируется в следующей программе.

Программа 3.7. Тип дня недели

```
// Файл TypeDayWeek.cpp

// Программа выводит тип дня недели (рабочий или выходной) по его номеру
```

```
#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   int day;                          // Номер дня недели
   cout << "Введите номер дня недели от 1 до 7 \n";
   cin >> day;
   cout << "Это ";
   switch(day){
      case 1: case 2: case 3:
      case 4: case 5: cout << "рабочий день\n";   break;
      case 6: case 7: cout << "выходной\n";       break;
      default: cout << "неверный номер\n";
   }
   system("pause");                  // Ждем нажатия клавиши
   return 0;
}
```

Далее приведены три диалога с программой.

```
Введите номер дня недели от 1 до 7
3
Это рабочий день

Введите номер дня недели от 1 до 7
7
Это выходной

Введите номер дня недели от 1 до 7
9
Это неверный номер
```

3.6. Операторы break и continue

Оператор `break` вызывает немедленный выход из циклов `for`, `while`, `do-while` и переключателя `switch`. Выход осуществляется только из одного самого внутреннего из нескольких охватывающих циклов или переключателей.

Оператор `continue` вызывает следующее повторение охватывающего его цикла `for`, `while`, `do-while` до окончания выполнения всех инструкций в теле цикла. Использование операторов `break` и `continue` иллюстрируется следующей программой.

Программа 3.8. Сумма положительных чисел

В программе в бесконечном цикле for вводятся числа, положительные суммируются, отрицательные числа игнорируются, а при поступлении нуля используется break для выхода из цикла.

```
// Файл SumPositive.cpp

// Подсчет суммы положительных чисел и их количества

#include <iostream>
#include <locale>
#include <cstdlib>
using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");
   int x, sum = 0;           // Вводимое число и сумма
   int i = 0;                // Количество введенных чисел
   int ipos = 0;             // Количество введенных положительных чисел
   cout << "\nВводите числа. 0 - выход:\n";
   for( ;  ; ){              // Бесконечный цикл
      cin >> x;
      i++;
      if(x < 0)              // Если число отрицательное,
         continue;           // переход к повторению цикла
      if(x == 0)
         break;              // Выход из цикла
      sum += x;              // Увеличение суммы
      ipos++;                // Увеличение счетчика положительных чисел
   }
   cout << "Введено " << i << " чисел,\n";
   cout << " из них " << ipos << " положительных.\n";
   cout << "Сумма положительных sum = " << sum << endl;
   system("pause");
   return 0;
}
```

Пример работы программы:

```
Вводите числа. 0 - выход:
1 2 3 -1 -4 6 7 -3 -9 0
Введено 10 чисел,
из них 5 положительных.
Сумма положительных  sum = 19
```