Введение в язык C++

Углич П. С.[1]

[1]Южный федеральный университет

Лекция 2

Outline

Цикл for

```
// форма записи оператора цикла for:
for (/*выражение1*/; /*выражение2*/; /*выражение3*/ )
{/*один оператор или блок операторов*/;}
```

- **выражение 1** — объявление (и) или инициализация, ранее объявленной, переменной-счетчика, которая будет отвечать за истинность условия в цикле for.
- **выражение 2** — это условие продолжения цикла for, оно проверяется на истинность.
- **выражение 3** — выражение 3 изменяет значение переменной-счетчика. Без выражения 3 цикл считается бесконечным

Примеры

```
for (int i = 0; i < 10; i++)
{
cout << i <<" ";
}

for (double i = 0; i < 1.0; i+=0.1)
{
cout << i << " " << sin(i) <<"\n";
}
```

Примеры

Таблица Пифагора

```
for (int i = 0; i < 10; i++)
{
cout << i <<" ";
}

for (double i = 0; i < 1.0; i+=0.1)
{
cout << i << " " << sin(i) <<"\n";
}
```

Цикл while

```
// форма записи цикла while
while (/*условие продолжения цикла while*/)
{
/*блок операторов*/;
/*управление условием*/;
}
```

Условие продолжения цикла должно быть истинно true, как только условие стало ложным, выполняется выход из цикла.

Пример

```
const double eps = 0.1e-10;
double s = 1.0;
double s1 = 1.0;
double x = 1.0;
int i = 1;
while (abs(s1) > eps)
{
s1 *= x / i;
s += s1;
i++;
}
cout << s << "\n";
```

Цикл do while

```
const double eps = 0.1e-10;
double s = 1.0;
double s1 = 1.0;
double x = 1.0;
int i = 1;
do
{
s1 *= x / i;
s += s1;
i++;
} while (abs(s1) > eps);
cout << s << "\n";
```

Пример do while

```
const double eps = 0.1e-10;
double s = 1.0;
double s1 = 1.0;
double x = 1.0;
int i = 1;
while (abs(s1) > eps)
{
s1 *= x / i;
s += s1;
i++;
}
cout << s << "\n";
```

Пример

Дана последовательность целых чисел (вводятся с клавиатуры), последний элемент которой — число 0. Найти сумму всех положительных элементов этой последовательности и количество ее отрицательных элементов.

```
int sum = 0;
int quantity = 0;
while (true)
{          int xx;
cin >> xx;
sum += xx;
if (xx<0) quantity++;
if (xx == 0) break; }
cout << "Сумма элементов " << sum <<
 "  Количество отрицательных элементов -  "
 << quantity <<"\n";
```

Пример

Когда оператор break выполняется в цикле, то досрочно прерывается исполнение оператора цикла, и управление передается следующему оператору после цикла.
Оператор continue используется только в циклах. В операторах for, while, do while, оператор continue выполняет пропуск оставшейся части кода тела цикла и переходит к следующей итерации цикла.

Функции — это блоки кода, выполняющие определенные операции.Если требуется, функция может определять входные параметры, позволяющие вызывающим объектам передавать ей аргументы.При необходимости функция также может возвращать значение как выходное. В идеальном случае имя функции должно четко описывать назначение функции.Следующая функция принимает от вызывающего ее объекта два целых числа и возвращает их сумму. a и b — параметры типа int.

```
int sum(int a, int b)
{
    return a + b;
}
```

```
// структура объявления функций не возвращающих значений
void /*имя функции*/(/*параметры функции*/)
 // заголовок функции
{
// тело функции
}

// структура объявления функций, возвращающих значения
/*возвращаемый тип данных*/ /*имя функции*/
(/*параметры функции*/) // заголовок функции
{
// тело функции
 return /*возвращаемое значение*/;
}
```

При объявлении функции необходимо указать:

- Возвращаемый тип, представляющий собой тип значения, возвращаемого функцией. Если возвращать значение не требуется, укажите void. Можно использовать возвращаемый тип auto, отдающий компилятору команду определять тип в соответствии с оператором return.
- Имя функции, которое должно начинаться с буквы или символа подчеркивания и не должно содержать пробелов.
- Список параметров, заключенный в скобки. В этом списке через запятую указывается нужное (возможно, нулевое) число параметров, задающих тип и, при необходимости, локальное имя, по которому к значениям можно получить доступ в теле функции.

inline-функции

Для маленьких функций накладные расходы на вызов функции значительно превышают вычисления, производимые внутри функции. Поэтому такую функцию можно сделать встраиваемой: при компиляции тело будет встроено на место ее вызова.

```
inline int add(int a, int b) {
    return a + b;
}

int a = 3;
int c = add(a, 5);    // int c = a + 5;
```

Передача аргумента в функцию по ссылке

```
void calcSquareAndPerimeter(double a, double b,
 double &S, double &P)
 {
    S = a * b;
    P = 2 * (a + b);
}
int main() {
    double SS = 0.0, PP = 0.0;
    calcSquareAndPerimeter(3.0, 4.0, SS, PP);
    cout << "Perimeter: " << PP
    << ", Square: " << SS << "\n";
}
```

Объявление функций

```
#include <iostream>
/*область 1 - определение функций до начала main()
место для объявления функций
функциям объявленным в этой области не нужны прототипы
*/
int main()
{
 return 0;
}
```

Объявление функций

```
#include <iostream>
// место для объявления функций
int main()
{
 return 0;
}
/*область 2 - объявление функций после main()
 место для определения функций
 */
```

Объявление функций

Объявления прототипов необходимы в том случае, когда программа имеет многомодульную структуру и определение функции располагается не в том файле, где она используется. Поэтому объявление функций является хорошей подготовкой к организации многомодульной структуры программы.

Пример

```
#include <iostream>
using namespace std; //использовать пространство имен std
/*объявления функций
сами функции описаны после функции main()
*/
void my_swap(double &a, double &b);
void my_swap(int &a, int &b);
int main() {
int k,m;
cout<<"enter two integer values:";
cin>>k>>m;
cout<<k<<" "<<m<<endl;
my_swap(k, m);
cout<<k<<" "<<m<<endl;
```

Пример

```
double a,b;
cout<<"enter two double values:";
cin>>a>>b;
cout<<a<<" "<<b<<endl;
my_swap(a,b);
cout<<a<<" "<<b<<endl;
return 0;
}
```

Пример

```
void my_swap(double &a, double &b) {
double r = a;
a=b;
b=r;
}
void my_swap(int &a, int &b) {
int r=a;
a=b;
b=r;
}
```

Перегрузка функций

Под перегрузкой функции понимается, определение нескольких функций (две или больше) с одинаковым именем, но различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом.

```
int print( char *s );                    // Print a string.
int print( double dvalue );              // Print a double.
int print( double dvalue, int prec );  // Print a double
//with a  given precision.
```

Компилятор самостоятельно выберет нужную функцию, анализируя только лишь сигнатуры перегруженных функций.

Шаблоны функций

Шаблоны функций похожи на шаблоны классов, но определяют семейство функций.С помощью шаблонов функций можно задавать наборы функций, основанных на одном коде, но действующих в разных типах или классах. Следующий шаблон функции меняет местами два элемента.

```
template <typename T>
void Swap(T a, T b)
{
T r = a;
a = b;
b = r;
} // конец шаблона функции
```

Шаблоны функций

Все шаблоны функций начинаются со слова template, после которого идут угловые скобки, в которых перечисляется список параметров. Каждому параметру должно предшествовать зарезервированное слово class или typename.

```
template <class T>
```

или

```
template <typename T>
```

или

```
template <typename T1, typename T2>
```

Шаблоны функций

Шаблон функции не позволяет разработчику менять местами объекты разных типов, поскольку компилятор во время компиляции знает типы параметров a и b.

```
int j = 10;
int k = 18;
CString Hello = "Hello, Windows!";
Swap( j, k );           //OK
Swap( j, Hello );       //error
```

Второй вызов Swap приводит к ошибке времени компиляции

Шаблоны функций

Для шаблона функции можно явно задавать аргументы.
Например:

```
// function_templates2.cpp
template<class T> void f(T) {}
int main(int j) {
   f<char>(j);   // Generate the specialization f(char).
   // If not explicitly specified, f(int) would be deduced.
}
```

Шаблоны функций

```
template <typename T>
void printArray(const T * array, int count)
{
    for (int ix = 0; ix < count; ix++)
        cout << array[ix] << "   ";
    cout << endl;
} // конец шаблона функции printArray
```

Аргументы по умолчанию

Во многих случаях функции имеют аргументы, которые используются настолько редко, что достаточно значения по умолчанию. В таких случаях возможность задания аргументов по умолчанию позволяет указывать только те аргументы функции, которые важны в конкретном вызове.
Пример:

```
int print( char *s );                    // Print a string.
int print( double dvalue );              // Print a double.
int print( double dvalue, int prec );
```

Аргументы по умолчанию

Во многих приложениях для аргумента prec можно задать разумное значение по умолчанию, что исключает необходимость наличия двух функций:

```
int print( char *s );
int print( double dvalue, int prec=2 );
```

При использовании аргументов по умолчанию обратите внимание на следующие моменты:

- Они должны быть последними аргументами.Поэтому следующий код недопустим:

  ```
  int print( double dvalue = 0.0, int prec );
  ```

- Переопределение аргумента по умолчанию в последующих объявлениях не допускается, даже если оно совпадает с оригиналом.

Аргументы по умолчанию

Следующий код недопустим

```
// Prototype for print function.
int print( double dvalue, int prec = 2 );

...

// Definition for print function.
int print( double dvalue, int prec = 2 )
{
...
}
```

В последующих определениях допускается добавлять дополнительные аргументы по умолчанию.

Локальные и глобальные переменные

Обычно глобальные переменные объявляются перед главной функцией, но можно объявлять и после функции main(), но тогда данная переменная не будет доступна в функции main().

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void example();
int variable = 48; // инициализация глобальной переменной
int main(int argc, char* argv[])
{       int variable = 12; // инициализация локальной перемен
        cout << "local  variable = " << variable << endl;
        example(); // запуск функции
        return 0; }
void example()
{        cout << "global variable = " << variable << endl; }
```

Локальные и глобальные переменные

В C++ существует такая операция, как разрешение области действия ::. Эта операция позволяет обращаться к глобальной переменной из любого места программы.

```
cout << "local  variable = " << ::variable << endl; /
```

Операция разрешения области действия ставится перед именем глобальной переменной, и даже, если есть локальная переменная с таким же именем, программа будет работать со значением, содержащимся в глобальной переменной.

Заголовочные файлы

Программа на языке C++ состоит из одной или нескольких функций, объявлений и определений глобальных объектов, подключения пространства имен и заголовочных файлов.
Среди функций должна быть одна и только одна с именем main().
Все функции, составляющие программу, могут быть расположены в одном файле — исходном модуле.
Реальные программы на C++, как правило, состоят из нескольких исходных модулей.
В случае вынесения функций в отдельный cpp-файл следует создавать соответствующий ему h-файл, содержащий объявления соответствующих функций.
Рекомендуется, чтобы имена cpp-файла и соответствующего ему h-файла совпадали.
Основное правило рекомендует ограничиться одними объявлениями.

Заголовочные файлы

```
typedef int(*IFnI)(int);
void my_map(int *a, int * b, int size, IFnI f);
int F1(int x);
int F2(int x);

template <typename T>
void Swap(T a, T b)
{
T r = a;
a = b;
b = r;
} // конец шаблона функции
```

Заголовочные файлы

Может оказаться, что заголовочный файл многократно включается в сложную программу.
В результате возникают ошибки, связанные с многократным определением. (Объявлений может быть сколько угодно, определение может быть только одно). В каждом заголовочном файле следует проверить, не был ли этот файл уже включен в проект. Это делается при помощи препроцессорного флага.

```
#ifndef HEADER_FLAG
#define HEADER_FLAG
//
необходимые
объявления
#endif    // HEADER_FLAG
```