Язык программирования C++

Лекция 4

Функции

- прототипы
- заголовочные файлы
- аргументы

Структура программ: функции

Пример функции

```
double coulombsLaw (double q1, double q2, double r) {
// Coulomb's law for the force acting on two point charges
// q1 and q2 at a distance r.  MKS units are used.

        double k = 8.9875e9;              //nt*m^2/coul^2
        return k * q1 * q2 / (r * r);
}

int main()  {
        cout << coulombsLaw(1.6e-19, 1.6e-19, 5.3e-11)
                << " newtons" << endl;
        return 0;
}
```

- первый компонент (double) – тип возвращаемого объекта
- второй – имя функции
- аргументы задаются в ( ), перед именами аргументов указываются их типы
- тело функции заключено в скобки { }
- оператор возврата может быть выражением, переменной или константой
- функция может не возвращать никакой объект, тогда ее тип void
- если у функции нет аргументов, то оставляется пустое место или ставится void

Прототипы функций

```
int main()  {
        cout << coulombsLaw(1.6e-19, 1.6e-19, 5.3e-11)
                << " newtons" << endl;
        return 0;
}

double coulombsLaw (double q1, double q2, double r) {
// Coulomb's law for the force acting on two point charges
// q1 and q2 at a distance r.  MKS units are used.

        double k = 8.9875e9;            //nt*m^2/coul^2
        return k * q1 * q2 / (r * r);
}
```

- C++ проверяет количество аргументов функции и их типы
- производит при необходимости стандартное преобразование типов
- проверяет тип возвращаемого объекта
- если эти проверки не проходят – возникает ошибка компиляции; пример на этом слайде как раз и выдаст ошибку

Прототипы функций

необходимо перед использованием функции дать информацию компилятору о ее аргументах и типе возвращаемого объекта
т.е. создать прототип функции

```
double coulombsLaw(double, double, double);

int main()  {
        cout << coulombsLaw(1.6e-19, 1.6e-19, 5.3e-11)
                << " newtons" << endl;
        return 0;
}

double coulombsLaw (double q1, double q2, double r) {
// Coulomb's law for the force acting on two point charges
// q1 and q2 at a distance r.  MKS units are used.

        double k = 8.9875e9;            //nt*m^2/coul^2
        return k * q1 * q2 / (r * r);
}
```

Прототипы функций и заголовочные файлы

- Для удобства разработки больших программ, их лучше разбивать на отдельные модули, которые можно независимо компилировать и отлаживать
- Отлаженные таким образом стандартные функции помещаются в библиотеки
- Пользовательские программы используют эти библиотеки («загружают» их)
- Но пользователь должен описать прототип библиотечной функции и декларировать его у себя в программе идентично тому, как это сделано внутри библиотеки
- Для обеспечения этой идентичности создаются заголовочные файлы

Заголовочные файлы

Заголовочный файл math.h :

```
double sqrt(double);
double sin(double);
double cos(double);
// и много много других
```

Библиотечный файл math.c :

```
#include <math.h>
double sqrt(double x) {
//  вычисление квадратного корня
        return result;
}
double sin(double x) {
// вычисление синуса
        return result;
}
```

Пользовательский файл user.cpp :

```
#include <math.h>


...
double x, y, z, r;
//
r = sqrt (x*x + y*y + z*z);
```

прототипы функций из заголовочных файлов используются при компиляции библиотек, и именно они же вставляются в программы пользователя, которые используют эти библиотеки

Аргументы функции по умолчанию

Можно определить значения аргументов, которые не указываются при вызове функции

```
#include <math.h>
double log_of (double x, double base = M_E);
// константа M_E определена в файле math.h
```

после этого можно записать

```
#include <my_header/logof.h>

x = log_of(y);          // натуральный логарифм
z = log_of(y, 10);          // десятичный логарифм
```

- все аргументы справа от аргумента по умолчанию должны тоже иметь значения по умолчанию
- при использовании первого значения по умолчанию остальные аргументы тоже должны использовать свои
- значение по умолчанию должно быть определено в вызывающей программе

Заголовочные файлы и С-препроцессор

В больших программах может получиться так, что заголовочный файл вставляется больше одного раза, а это недопустимо и ведет к ошибке двойного определения одного и того же объекта или объектов.
Специальный метод препроцессора С позволяет это избежать

```
#ifndef MATH_H
#define MATH_H
double sqrt(double x);
double sin(double x);
double cos(double x);
// и много много других
#endif   // MATH_H
```

- препроцессор строит временный файл для компилятора
- MATH_H – макропеременная препроцессора по договоренности всегда пишется большими буквами
- #ifndef #define #endif – служебные директивы препроцессора
- большинство заголовочных файлов (особенно системных) используют этот метод

Перегрузка имени функции

В языке C разные функции, выполняющие одинаковые действия, но с объектами разного типа, имели разные имена

```
int          iabs(int);
float         abs(float);
double   dabs(double);
```

В языке C++ появилась возможность использовать одно и то же имя

```
int         abs(int i);
float       abs(float x);
double   abs(double y);
```

функции внутренне определяются не только по имени, но и по типу возвращаемого объекта, по количеству и типам аргументов

Пространство имен

Для того, чтобы избежать возможного пересечения имен функций из разных библиотек, в языке C++ было придумано пространство имен

```
#ifndef MATH_H
#define MATH_H
namespace std {
  double sqrt(double x);
  double sin(double x);
  double cos(double x);
// и много много других
}   //  end  namespace
#endif   // MATH_H
```

поэтому эти функции надо использовать так:

```
#include <math.h>

y = std::sin (x);
// или
using std::sin;
y = sin (x);
// или
using namespace std;
y = sin (x);
```