Лекция 2

Отличия языка C++ от языка C

Ввод и вывод

В языке C++ ввод и вывод осуществляется через потоки, но , в отличие от языка C, используется объектно-ориентированный подход:

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

int main(int argc, char* argv[])
{
  int x = 0, y = 0;
  cout << "Введите X: ";
  cin >> x;
  cout << "Введите Y: ";
  cin >> y;
  cout << "Сумма " << x << '+' << y << " = " << x+y << '\n';
  return 0;
}
```

Создание новых типов

В языке C++ объявление новых типов (enum, struct, union, class) осуществляется без использования оператора **typedef**:

```
struct Student{
  char fio[3][16];
  int kurs;
  float rate;
};

Student st = {”Иванов”,”Иван”,”Иванович”,1,7.0};
```

Создание новых типов

```
#include <iostream>
using namespase std;
int main(int argc, char* argv[])
{
  struct Student{
    char fio[3][16];
    int kurs;
    float rate;
  };

  Student st;

  cout << "Введите в формате \” Имя Фамилия Отчество курс успеваемость\“\n";
  cin >> st.fio[0] >> st.fio[1] >> st.fio[2] >> st.kurs >> st.rate;

  cout << “Имя: " << st.fio[0] << ' ' << st.fio[1] << ' ' << st.fio[2] << '\n';
  cout << “Курс: " << st.kurs << '\n';
  cout << “Успеваемость: " << st.rate << '\n';

  return 0;
}
```

Объявление функций без параметров

В языке C++ при объявлении функции без параметров допускается оставлять пустые скобки, не указывая ключевое слово void, как это делается в языке C.

```
#include <iostream>
using namespase std;

void Func();
int main(int argc, char* argv[])
{
  Func();
  return 0;
}
void Func()
{
  cout << "Функция без параметров!\n";
}
```

Встроенные функции

В языке C++ (а также в языке C стандарта C99) допускается создавать встроенные функции.

```
#include <iostream>
using namespase std;

inline double sqr(double x) { return x * x; }

int main(int argc, char* argv[])
{
  double x = 0.0, y = 0.0;
  cout << "Enter X: ";
  cin >> x;
  y = sqr(x);
  cout << x << "^2 = " << y << '\n';
  return 0;
}
```

Параметры-ссылки

В языке C++ реализован механизм передачи параметров по ссылке. Для этого используется описание параметра в виде:

тип &имя

Пример:

```
#include <iostream>
using namespase std;

int SqrByValue(int v) { return v*v; }
void SqrByPointer(int *v) { *v *= *v; }
void SqrByReference(int &v) { v *= v;}

int main(int argc, char* argv[])
{
  int x = 2, y = 3, z = 4;
  cout << "Перед SqrByValue: " << x << '\n';
  x = SqrByValue(x);
  cout << "После SqrByValue: " << x << '\n';

  cout << "Перед SqrByPointer: " << y << '\n';
  SqrByPointer(&y);
  cout << "После SqrByPointer: " << y << '\n';

  cout << "Перед SqrByReference: " << z << '\n';
  SqrByReference(z);
  cout << "После SqrByReference: " << z << '\n';
  return 0;
}
```

Параметры-ссылки

При передаче в параметрах функции большого значения (структура) целесообразно использовать передачу константной ссылки:

```
#include <iostream>
using namespase std;

struct Student{
  char fio[3][16];
  int kurs;
  float rate;
};

void PrintStudent(const Student &s)
{
  cout << “Имя: " << s.fio[0] << ' ' << s.fio[1] << ' ' << s.fio[2] << '\n';
  cout << “Курс: " << s.kurs << "\nУспеваемость: " << s.rate << '\n';
}

int main(int argc, char* argv[])
{
  Student st = {{"Иванов", "Иван", "Иванович"},3,6.5};
  PrintStudent(st);
  return 0;
}
```

Псевдонимы

В языке C++ допускается объявлять псевдонимы переменных используя синтаксис ссылок:

```
int x = 10;
int &y = x;
cout << "Y=" << y << endl; //вывод: 10
x += 10;
cout << "Y=" << y << endl; //вывод: 20
```

Динамическое распределение памяти

Выделение памяти:
указатель = **new** тип;

Освобождение памяти:
delete указатель;

Примеры:

```
int *ptr = new int;
cin >> *ptr;
cout << "Value: " << *ptr << endl;
delete ptr;

ptr = new int (10);
cout << "Value: " << *ptr << endl;
delete ptr;
```

Динамическое распределение памяти

Выделение памяти под массив:
указатель = **new** тип[размер];

Освобождение памяти:
delete [] указатель;

Пример:

```
int *ptr = new int [10];

for(int i=0;i<10;i++) cin >> ptr[i];

for(int i=0;i<10;i++) cout << ptr[i] << ' ';
cout << endl;

delete [] ptr;
```

Параметры по умолчанию

В языке C++ можно описывать функции с параметрами по умолчанию. Если при вызове функции значение данного параметра не указано, то используется значение по умолчанию.

Пример:

```
double Volume(double l, double =1.0, double =1.0);

int main(int argc, char* argv[])
{
  cout << "Объем: " << Volume(2,3,4) << endl;  //вывод: 24
  cout << "Объем: " << Volume(2,3) << endl;    //вывод: 6
  cout << "Объем: " << Volume(2) << endl;      //вывод: 2
  return 0;
}

double Volume(double l, double w, double h)
{
  return l*w*h;
}
```

Унарная операция разрешения области действия

Пример в «стиле» языка C:

```
int value = 10;

int main(int argc, char* argv[])
{
  int value = 5;
  cout << "Значение: " << value << endl;
  return 0;
}
```

На экране: Значение: 5

Унарная операция разрешения области действия

Пример на языке C++:

```
int value = 10;

int main(int argc, char* argv[])
{
  int value = 5;
  cout << "Локальное значение: " << value << endl;
  cout << "Глобальное значение: " << ::value << endl;
  return 0;
}
```

На экране:
Локальное значение: 5
Глобальное значение: 10

Перегрузка функций

В языке C++ допускается перегрузка функций – возможность использования одного и того же идентификатора для именования нескольких функций.

Пример:

```
double Square(double);
double Square(double,double);

int main(int argc, char* argv[])
{
  cout << "Площадь окружности: " << Square(5) << endl;
  cout << "Площадь прямоугольника: " << Square(2,6) << endl;
  return 0;
}

double Square(double r) { return 3.1415*r*r; }
double Square(double a, double b) { return a*b; }
```

Перегрузка функций

Сигнатура функций:
@имя$qпараметры

Примеры:
double square(double) @square$qd
double square(double,double) @square$qdd

Шаблоны функций

Все определения шаблонов функций начинаются с ключевого слова **template**, за которым следует список формальных параметров шаблона функции, заключенный в угловые скобки (< и >). Каждому формальному параметру предшествует ключевое слово **class**. Эти формальные параметры используются подобно встроенным типам или типам, определенными пользователем, для задания типов параметров функции, задания типа возвращаемого значения и объявления переменных внутри функций. Далее следует определение шаблона, которое не отличается от определения функции.

Шаблоны функций

```
template <class T>
void printArray(T* array, const int n)
{
  for(int i=0;i<n;i++) cout << array[i] << ' ';
  cout << endl;
}

int main(int argc, char* argv[])
{
  const int iCount = 5, fCount = 7;
  int iArr[iCount] = {1,2,3,4,5};
  float fArr[fCount] = {1.1,2.2,3.3,4.4,5.5,6.6,7.7};

  cout << "Целочисленный массив: ";
  printArray(iArr,iCount);

  cout << "Вещественный массив: ";
  printArray(fArr,fCount);

  return 0;
}
```