# Введение в язык C++

Углич П. С.[1]

[1]Южный федеральный университет

Лекция 1

# Outline

# Простейшая программа

Каждая программа на C++ состоит из одной или более функций. Среди них обязательно должна присутствовать главная функция main, которую и вызывает операционная система при запуске программы. Простейший пример:

```
int main() {
    return 0;
}
```

Значение, возвращаемое main, используется ОС для определения успешности завершения программы. Если не ноль, то в программе произошла ошибка. int — тип возвращаемого значения.

```
#include <iostream>   // Заголовочный файл
int main() {
    std::cout << "Hello world!" << std::endl;
}
```

#include — это директива «препроцессору», которая сообщает компилятору поместить код из заголовочного файла iostream в нашу программу перед тем как создать исполняемый файл.

Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:

- cin — объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;
- cout — объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;
- cerr — объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево («), а ввод — с помощью оператора сдвига вправо (»).
Назначение операторов легче запомнить, если считать, что каждый «указывает» в сторону перемещения данных.
Например,

```
>> x
```

перемещает данные в x, а

```
<< x
```

перемещает данные из x.
endl — это манипулятор вывода, который вставляет в выходной поток символ перехода на новую строку, а затем сбрасывает буфер объекта ostream.

Пространство имен — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных, и т. д.). Все типы и функции стандартной библиотеки C++ объявлены в пространстве имен std или в пространстве имен, вложенном в std.
Строка using namespace std; сообщает компилятору, что нужно использовать группу функций, которые являются частью стандартной библиотеки std.

```
#include <iostream>
using namespace std;
//using std::cout;
int main() {
    cout << "Hello world!"<<endl;
}
```

или

```
#include <iostream>
int main() {
   using namespace std;
    cout << "Hello world!"<<endl;
}
```

## Переменные и типы данных

**b**ool — логический тип данных, переменная типа bool занимает 1 байт и принимает значения от нуля до 255, при этом ноль воспринимается как false, а всякое значение, отличное от нуля — как true.
**c**har — целочисленный тип данных, который используется для представления символов. То есть, каждому символу соответствует определенное число из диапазона [0; 255].

```
#include <iostream>   // Заголовочный файл
#include <clocale>
int main() {
using namespace std;
// вызов функции настройки локали
setlocale(LC_CTYPE, "rus");
cout << "Кириллица в консоли\n";
}
```

Расширенный символьный тип

```
wchar_t ch1[] = L"hello";
wcout << ch1 << endl;
```

Целочисленные типы данных:

- **short**
- **unsigned short**
- **int**
- **unsigned int**
- **__int8, __int16, __int32, __int64**
- **long**
- **unsigned long**
- **long long**

Числа с плавающей запятой

- **float**
- **double**
- **long double**

Определение констант

```
const int n=10;
```

const *название типа* **имя переменной** = значение;

Автоопределение типа

```
auto i = 42;          // i - int
auto l = 42LL;        // l - long long
auto p = new foo(); // p - foo*
```

# Арифметические операторы

- + — сложение;
- - — вычитание;
- * — умножение;
- / — деление;
- % — остаток от деления.

# Побитовые операторы

- `expression1 & expression2;// побитовое И`
- `expression1 ^ expression2;// побитовое исключающее ИЛИ;`
- `expression1 | expression2;// побитовое ИЛИ;`
- `expression1 << n;// сдвиг влево на n позиций;`

```
unsigned short aa = 0xFFFF;      // pattern 1111 ...
unsigned short bb = 0xAAAA;      // pattern 1010 ...*/
cout << hex << (aa & bb) << endl;
// prints "aaaa", pattern 1010 ...
cout << hex << (aa ^ bb) << endl;
// prints "aaaa", pattern 1010 ...
cout << hex << (aa | bb) << endl;
   // prints "aaaa", pattern 1010 ...
```

# Логические операторы

- && — И;
- || — ИЛИ;
- ! — НЕ;

Перед вычислением оба операнда неявно преобразуются в тип bool; результат также имеет тип bool.

# Операторы присваивания

```
expression assignment-operator expression
assignment-operator : one of
   =   *=   /=   %=   +=   -=   <<=   >>=   &=   ^=   |=
```

- = — присваивание; Простой оператор присваивания (=) сохраняет значение второго операнда в объекте, указанном первым операндом. Если оба объекта имеют арифметические типы, перед сохранением значения правый операнд преобразуется к типу левого.
- += — Сложение значения первого операнда со значением второго операнда; сохранение результата в объект, указанный первым операндом;
- -= — присваивание-вычитание;
- *= — присваивание-умножение;
- /= — присваивание-деление;
- %= — присваивание-остаток от деления;

# Операторы инкремента и декремента

```
i++; i--; // постфиксная форма

++i; --i;  // префиксная форма
```

В результате применения оператора инкремента (++) значение операнда увеличивается на одну единицу соответствующего типа, а в результате применения постфиксного оператора декремента значение операнда уменьшается на одну единицу соответствующего типа.
Постфиксная операция инкремента или декремента выполняется после вычисления операнда. В префиксной форме инкремент или декремент выполняется до использования значения при вычислении выражения, поэтому значение выражения отличается от значение операнда.

# Операторы отношения и равенства

```
expression < expression
expression > expression
expression <= expression
expression >= expression
expression == expression
expression != expression
```

Оба операнда операторов отношения должны быть арифметического типа или типа указателя. Они возвращают значения типа bool. Значение false (0) возвращается, если отношение в выражении ложно; в противном случае возвращается значение true (1). Оператор "равно"(==) возвращает значение true (1), если оба операнда имеют одинаковые значения. Оператор "не равно"(!=) возвращает значение true, если операнды имеют неравные значения.

# Условный оператор

```
if (/*проверяемое условие*/)
{
   /*тело оператора выбора 1*/;
} else
      {
         /*тело оператора выбора 2*/;
      }
```

Тернарный условный оператор

```
// "условие" ? "выражение 1" : "выражение 2";
a > b ? cout << a : cout << b;
// если a > b, то выполняется cout << a,
// иначе выполняется cout << b
 cout << "y = " <<
 (x < 0 ? x : (x >= 0) && (x < 30) ? 0 : x * x  ) << endl;
```

# Оператор выбора

```
// форма записи оператора множественного выбора switch
switch (/*переменная или выражение*/)
 {
 case /*константное выражение1/*:
 { /*группа операторов*/; break; }
 case /*константное выражение2*/:
 { /*группа операторов*/; break; }
 //.    .    .
 default: { /*группа операторов*/; }
 }
```