

# Программирование на языке C++

## Лекция 1

Введение в синтаксис C++

Александр Смаль

# Типы данных

- Целочисленные:

1. `char` (символьный тип данных)
2. `short int`
3. `int`
4. `long int`

Могут быть беззнаковыми (`unsigned`).

- $-2^{n-1} \dots (2^{n-1} - 1)$  ( $n$  — число бит)
  - $0 \dots (2^n - 1)$  для `unsigned`
- Числа с плавающей точкой:
    1. `float`, 4 байта, 7 значащих цифр.
    2. `double`, 8 байт, 15 значащих цифр.
  - Логический тип данных `bool`.
  - Пустой тип `void`.

# Литералы

- Целочисленные:
  1. 'a' — код буквы 'a', тип `char`,
  2. 42 — все целые числа по умолчанию типа `int`,
  3. 1234567890L — суффикс 'L' соответствует типу `long`,
  4. 1703U — суффикс 'U' соответствует типу `unsigned int`,
  5. 2128506UL — соответствует типу `unsigned long`.
- Числа с плавающей точкой:
  1. 3.14 — все числа с точкой по умолчанию типа `double`,
  2. 2.71F — суффикс 'F' соответствует типу `float`,
  3. 3.0E8 — соответствует  $3.0 \cdot 10^8$ .
- `true` и `false` — значения типа `bool`.
- Строки задаются в двойных кавычках: "Text string".

# Переменные

- При определении переменной указывается её тип. При определении можно сразу задать начальное значение (инициализация).

```
int      i = 10;  
short    j = 20;  
bool     b = false;  
  
unsigned long l = 123123;  
  
double x = 13.5, y = 3.1415;  
float  z;
```

- Нужно всегда инициализировать переменные.
- Нельзя определить переменную пустого типа `void`.

# Операции

- Оператор присваивания: `=`.
- Арифметические:
  1. бинарные: `+` `-` `*` `/` `%`,
  2. унарные: `++` `--`.
- Логические:
  1. бинарные: `&&` `||`,
  2. унарные: `!`.
- Сравнения: `==` `!=` `>` `<` `>=` `<=`.
- Приведения типов: `(type)`.
- Сокращённые версии бинарных операторов: `+=` `-=` `*=` `/=` `%=`.

```
int i = 10;
i = (20 * 3) % 7;

int k = i++;
int l = --i;

bool b = !(k == 1);

b = (a == 0) ||
    (1 / a < 1);

double d = 3.1415;
float f = (int)d;

// d = d * (i + k)
d *= i + k;
```

# Инструкции

- Выполнение состоит из последовательности *инструкций*.
- Инструкции выполняются одна за другой.
- Порядок вычислений внутри инструкций не определён.

```
/* unspecified behavior */  
int i = 10;  
i = (i += 5) + (i * 4);
```

- Блоки имеют вложенную область видимости:

```
int k = 10;  
{  
    int k = 5 * i;    // не видна за пределами блока  
    i = (k += 5) + 5;  
}  
k = k + 1;
```

# Условные операторы

- Оператор `if`:

```
int d = b * b - 4 * a * c;  
if ( d > 0 ) {  
    roots = 2;  
} else if ( d == 0 ){  
    roots = 1;  
} else {  
    roots = 0;  
}
```

- Тернарный условный оператор:

```
int roots = 0;  
if (d >= 0)  
    roots = (d > 0 ) ? 2 : 1;
```

# Циклы

- Цикл `while`:

```
int squares = 0;
int k = 0;
while ( k < 10 ) {
    squares += k * k;
    k = k + 1;
}
```

- Цикл `for`:

```
for ( int k = 0; k < 10; k = k + 1 ) {
    squares += k * k;
}
```

- Для выхода из цикла используется оператор `break`.



# Функции

- В сигнатуре функции указывается тип возвращаемого значений и типы параметров.
- Ключевое слово `return` возвращает значение.

```
double square(double x) {  
    return x * x;  
}
```

- Переменные, определённые внутри функций, — *локальные*.
- Функция может возвращать `void`.
- Параметры передаются по значению (копируются).

```
void strange(double x, double y) {  
    x = y;  
}
```

# Макросы

- Макросами в C++ называют инструкции препроцессора.
- Препроцессор C++ является самостоятельным языком, работающим с произвольными строками.
- Макросы можно использовать для определения функций:

```
int max1(int x, int y) {  
    return x > y ? x : y;  
}  
  
#define max2(x, y)    x > y ? x : y  
  
a = b + max2(c, d);           // b + c > d ? c : d;
```

- Препроцессор “не знает” про синтаксис C++.

# Макросы

- Параметры макросов нужно оборачивать в скобки:

```
#define max3(x, y)    ((x) > (y) ? (x) : (y))
```

- Это не избавляет от всех проблем:

```
int a = 1;  
int b = 1;  
int c = max3(++a, b);  
// c = ((++a) > (b) ? (++a) : (b))
```

- Определять функции через макросы — плохая идея.
- Макросы можно использовать для условной компиляции:

```
#ifdef DEBUG  
    // дополнительные проверки  
#endif
```

# Ввод-вывод

- Будем использовать библиотеку `<iostream>`.

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

- Ввод:

```
int a = 0;
int b = 0;
cin >> a >> b;
```

- Вывод:

```
cout << "a + b = " << (a + b) << endl;
```

# Простая программа

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

int main ()
{
    int a = 0;
    int b = 0;

    cout << "Enter a and b: ";
    cin >> a >> b;

    cout << "a + b = " << (a + b) << endl;

    return 0;
}
```