# Переменные, ввод-вывод, операции

Denis Bakin

# Введение

## План занятия

- Особенности C++
- Первая программа
- Переменные и типы
- Ввод-вывод
- Арифметика и приведения типов
- Частые задачи (гипотенуза, разряды)
- Представление чисел (two's complement)
- Практика и подводные камни

## C++ кратко

- Статическая типизация

## C++ кратко

- Статическая типизация
- Производительность и контроль памяти

## C++ кратко

- Статическая типизация
- Производительность и контроль памяти
- Совместимость с C

## C++ кратко

- Статическая типизация
- Производительность и контроль памяти
- Совместимость с C
- Компилируемый

- Компилируемый

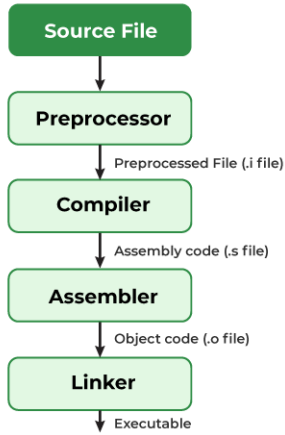


Figure 1: Процесс компиляции C++

# Первая программа

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!\n";
}
```

Ключевые элементы:

- #include
- Точка входа: int main()
- Поток вывода: std::cout
- Перенос строки: '\n'

# Переменные и типы

## Переменная

- Имя + тип + (значение)
- Память выделяется на этапе компиляции (для статически известных объектов)
- Инициализацию не пропускать

```
int x = 0;
double pi = 3.14159;
bool ok = true;
char letter = 'A';

std::cout << "PI equals to " << pi << std::endl;
```

# Переменные и типы

## Создание переменной

```
int main() {
    int a, b, c; // объявление без инициализации
    a = 10; // оператор присваивания

    // вариант с одновременной инициализацией является предпочтительным
    int new_num_1 = 10; // создаем переменную со значением 10
    int new_num_2{10}; // альтернатива
}
```

# Переменные и типы

## Неопределенное поведение (undefined behavior)

```
int x;        // неинициализировано (мусор)
std::cout << x << '\n'; // неопределённое поведение
```

- стандарт языка не определяет поведение программы
- всегда нужно инициализировать переменные корректно
- для поиска ошибок можно использовать флаги компиляции: -Wall -Wextra

## Переменные и типы

### Типы (основные)

```
// #include <string> должен быть написан выше для корректной работы с std::string


char c = '1';                  // символ
std::string s = "text";        // строка, не является встроенным типом, состоит из символов (c


bool b = true;                 // boolean, булевая, логическая переменная, принимает значения t

// целые числа
int i = 42;                    // integer, целое число (как правило, 4 байта)
short int si = 17;             // короткое целое (занимает 2 байта)
long li = 12321321312;         // длинное целое (как правило, 8 байт)

// числа с плавающей точкой
float f = 2.71828;             // дробное число с плавающей запятой (4 байта)
double d = 3.141592;           // дробное число двойной точности (8 байт)
long double ld = 1e15;         // длинное дробное (как правило, 16 байт)
```

# Переменные и типы

## Размеры (для x86-64)

- int: 4 байта
- long: 8 байт (Linux)
- long long: 8 байт
- float: 4 байта
- double: 8 байт
- long double: 16 байт (часто)

Проверка:

```
std::cout << sizeof(int) << '\n';
```

## Явные литералы

```
auto a = 10;       // int
auto b = 10u;      // unsigned int
auto c = 10LL;     // long long
auto d = 3.14;     // double
auto e = 3.14f;    // float
```

# Область видимости (scope)

## Пример

```
bool global_flag = false;

int main() {
    int x = 10;
    {
        int y = 20;
        std::cout << x << ' ' << y << '\n';
    }
    // y здесь недоступен
}
```

- Жизнь переменной ограничена блоком {}
- Минимизировать область видимости

# Ввод-вывод

## Потоки C++

```
#include <iostream>
#include <string>

int main() {
    std::string name;  // объявляем переменную name
    std::cout << "What is your name?\n";
    std::cin >> name;  // считываем её значение с клавиатуры
    std::cout << "Hello, " << name << "!\n";

    // объявляем нужные нам переменные
    // не инициализируем, потому что сразу будем в них читать значения
    int age, height;
    std::cout << "What is your age?\n";
    std::cin >> age;
    std::cout << "How tall are you?" << std::endl;
    std::cin >> height;

    std::cout << "Hm, it seems " << name << " is " <<
            height << "cm tall at " << age << " years old!\n";
}
```

## Строка целиком

```
std::string line;
std::getline(std::cin, line);
```

Важно:

- Если перед этим был ввод через operator > > — возможно остался '\n'
- Решение: `std::cin.ignore();`

## Форматированный ввод/вывод (C)

```
#include <cstdio>

int x;
std::scanf("%d", &x);
std::printf("x = %d\n", x);
```

Использовать осторожно: - Смешивание с iostream ☒ возможны проблемы буферизации - Для строк предпочитать std::string + std::getline

# Арифметика и приведения

## Основные операции

- +, -, *, /, %
- Целочисленное деление: 7 / 3 = 2
- Остаток: 7 % 3 = 1

```
int a = 7, b = 3;
int q = a / b;    // 2
int r = a % b;    // 1
```

# Арифметика и приведения

## Приведения

```
// целочисленное и точное деление
int c = 6, d = 4;
c / d; // 1
static_cast<float>(c) / d; // 1.5
1. * c / d; // 1.5
```

1. `static_cast<T>(object)` приводит объект к типу `T` , если это возможно
2. `1. * c / d` — сначала выполнится умножение (`float * int = float`), затем точно деление, так как в нем участвует `float` : `float / int = float`

# Арифметика и приведения

## Инкремент / декремент

```
#include <iostream>

int main() {
    int a = 2;
    ++a;
    std::cout << a << '\n'; // 3
    std::cout << ++a << '\n'; // 4
    std::cout << a++ << '\n'; // 4
    std::cout << a << '\n'; // 5
}
```

В случае, когда на программу выбор оператора не повлияет, лучше выбрать префиксный инкремент

# Арифметика и приведения

## Сокращённые формы

```
int x = 10;
x += 5;
x -= 2;
x *= 3;
x /= 4;
x %= 3;
```

# Задача: гипотенуза

**Задача:**
Даны два целых числа $a$ и $b$. Найдите гипотенузу прямоугольного треугольника с катетами $a$ и $b$.

**Входные данные:**
В двух строках вводятся два целых положительных числа, не превышающих $1000$.

**Выходные данные:**
Выведите длину гипотенузы.

# Задача: гипотенуза

Решение

Формула для гипотенузы:

$$c = \sqrt{a^2 + b^2}$$

# Задача: гипотенуза

## Пример кода на C++

```cpp
#include <iostream>
#include <cmath>

int main() {
    int a, b;
    std::cin >> a >> b;
    float hypot = std::sqrt(a * a + b * b);
    std::cout << hypot << '\n';
}
```

Уведомление

**Примечание:**
Для вычисления квадратного корня используется функция `std::sqrt` из библиотеки `<cmath>`.

# Задача: гипотенуза

## Точность вывода

```
#include <iomanip>
// ...
std::cout << std::fixed << std::setprecision(6) << c << '\n';
```

- `std::fixed` — фиксированный формат (без экспоненты): 123.456000 вместо 1.23456e+02
- `std::setprecision(6)` — 6 знаков после запятой

# Задача о разрядах числа

## Позиционные системы счисления

Любое число $N$ в системе счисления с основанием $d$ записывается как:

$$N = b_m \cdot d^{m-1} + b_{m-1} \cdot d^{m-2} + ... + b_2 \cdot d^1 + \underbrace{b_1 \cdot d^0}_{=b_1}$$

где $b_i$ — цифры числа, $0 \leq b_i < d$.

# Задача о разрядах числа

## Получение цифры в разряде

Чтобы получить цифру в разряде $k$ (отсчитывая с конца, $k = 0$ — единицы), используем:

$$\text{digit}_k = \left( \left\lfloor \frac{N}{d^k} \right\rfloor \right) \bmod d$$

Проще говоря, “убираем” деление разряды числа, а затем берем последнюю цифру – стоящую в новом разряде единиц

# Задача о разрядах числа

## Пример: десятичная система

Пусть $N = 12345$.

- Единицы: $12345 \bmod 10 = 5$
- Десятки: $\left\lfloor \frac{12345}{10} \right\rfloor \bmod 10 = 1234 \bmod 10 = 4$
- Сотни: $\left\lfloor \frac{12345}{100} \right\rfloor \bmod 10 = 123 \bmod 10 = 3$

# Задача о разрядах числа

Пример кода

```cpp
#include <iostream>
#include <cmath>


int main() {
    int num = 12345;
    std::cout << num % 10 << '\n';        // 5
    std::cout << num / 10 % 10 << '\n';   // 4
    std::cout << num / 100 % 10 << '\n'; // 3
}
```

# Представление целых чисел

## Побитово (целые)

- Память = фиксированное количество бит
- Беззнаковые: диапазон $0...2^k - 1$
- Знаковые: two's complement (двойное дополнение)

# Представление целых чисел

## Sign-magnitude (знак + модуль)

Идея: старший бит = знак, остальные = модуль

Проблемы:

- Два нуля ($+0$ и $-0$)
- Отдельная логика для сложения

| Биты (8) | Sign-magnitude | Беззнаковое |
|---|---|---|
| 0000 0001 | +1 | 1 |
| 1000 0001 | -1 | 129 |
| 1000 0000 | -0 | 128 |

# Представление целых чисел

## Сравнение (фрагмент)

| Бинарное | Sign-mag | Unsigned |
|---|---|---|
| 0111 1111 | +127 | 127 |
| 1000 0001 | -1 | 129 |
| 1111 1111 | -127 | 255 |

Вывод: арифметика неудобна для компьютера

# Представление целых чисел

## Two's complement (двоично-дополнительный код)

Алгоритм для $-x$ (некоторого целого числа с минусом):

1. Записать $x$ в двоичном виде
2. Инвертировать биты
3. Прибавить $1$

Свойства:

- Один ноль
- Сложение / вычитание как в беззнаковом (игнорируя переполнение)
- Диапазон: $-2^{(k-1)}...2^{(k-1)}-1$

# Представление целых чисел

Пример: получить -6 (4 бита)

$+6 = 0110$

Инвертируем $\rightarrow 1001$

$+1 \rightarrow 1010$

| Шаг | Биты |
|---|---|
| +6 | 0110 |
| Инверсия | 1001 |
| +1 | 1010 |
| Результат (-6) | 1010 |

# Представление целых чисел

## Проверка веса битов (4 бита)

(Старший бит = -8)

| Бит | 1 | 0 | 1 | 0 |
|---|---|---|---|---|
| Вес | -8 | 4 | 2 | 1 |

1010 = -8 + 2 = -6

# Представление целых чисел

## Полная таблица (4 бита)

| Биты | Значение | Биты | Значение |
|---|---|---|---|
| 0000 | 0 | 1000 | -8 |
| 0001 | 1 | 1001 | -7 |
| 0010 | 2 | 1010 | -6 |
| 0011 | 3 | 1011 | -5 |
| 0100 | 4 | 1100 | -4 |
| 0101 | 5 | 1101 | -3 |
| 0110 | 6 | 1110 | -2 |
| 0111 | 7 | 1111 | -1 |

# Представление целых чисел

Быстро получить $-x$ в коде

```
int x = 6;
int neg = (~x + 1); // двоично-дополнительное -x
std::cout << neg << '\n'; // -6
```

# Представление целых чисел

## Ключевые свойства

- Арифметика совпадает с беззнаковой по битам (модуль 2^k)
- Знаковое переполнение = undefined behavior (UB)
- Симметрия нарушена: |MIN| > MAX (например int: -256 .. 255)

# Итоги

- Базовый синтаксис и структура программы
- Переменные, типы, области видимости
- Потоки ввода-вывода
- Арифметика и преобразования
- Извлечение цифр
- Представление целых чисел