

# **Низкоуровневые средства C++ для работы с памятью (продолжение)**

Курс «Разработка ПО систем управления»

Кафедра управления и информатики НИУ «МЭИ»

Весна 2018 г.

# Операторы `static_cast`, `reinterpret_cast`

`static_cast <type> (object)`

`static_cast` осуществляет явное приведение типа. Создан для выполнения всех видов преобразований, разрешенных компилятором.

Поддерживается преобразование численных типов, указателей и ссылок по иерархии наследования как вверх, так и вниз. Проверка производится на уровне компиляции, так что в случае ошибки сообщение будет получено в момент сборки приложения или библиотеки.

`reinterpret_cast <type> (object)`

`reinterpret_cast` - осуществляет приведение типов без проверки - непосредственное указание компилятору считать биты и преобразовать их в другой тип, не обязательно совместимый.

Применяется только в случае полной уверенности программиста в собственных действиях.

# Побитовые операции

&	И	a	1	0	1	0	1	0	1	0	0xAA
	ИЛИ	b	0	0	0	0	1	1	1	1	0x0F
^	исключающее ИЛИ	a & b	0	0	0	0	1	0	1	0	0x0A
<<	сдвиг влево	a   b	1	0	1	0	1	1	1	1	0xAF
>>	сдвиг вправо	a ^ b	1	0	1	0	0	1	0	1	0xA5
~	НЕ	a << 1	0	1	0	1	0	1	0	0	0x54
		b >> 2	0	0	0	0	0	0	1	1	0x03
		~b	1	1	1	1	0	0	0	0	0xF0



По мотивам [слайдов](#)  
Бьярне Страуструпа.



По мотивам [слайдов](#)  
Бьярне Страуструпа.

# Битовые флаги

Если установлен этот бит, файл можно читать.

- `uint8_t constexpr CAN_READ = 04; // 0b'100`  
`uint8_t constexpr CAN_WRITE = 02; // 0b'010`  
`uint8_t constexpr CAN_EXECUTE = 01; // 0b'001`
- 
- Разные биты!

- Задание набора флагов логическим «ИЛИ»:

```
uint8_t CAN_EVERYTHING =  
    CAN_READ | CAN_WRITE | CAN_EXECUTE;  
// == 04 | 02 | 01 == 0b'100 | 0b'010 | 0b'001 == 0b'111 == 07
```

- Проверка наличия флага логическим «И»:

```
uint8_t permissions = 05;  
if (permissions & CAN_READ) { ... }  
// 05 & 04 == 0b'101 & 0b'100 == 0b'100 != 0 → true
```

# Битовые маски и сдвиги

- Задача:
- `uint32_t full = 0xFFFF; //0b'1111'1111'1111'1111`
- получить биты 4...15 из `uint32_t`.

✓Решение:

- сдвинуть нужные биты к началу числа (в 0...11);
- `full << 4`
- оставить только нужные биты (остальные обнулить).
- `(full << 4) & 0b'1111'1111'1111'0000 // 0xFFFF0`

- Задача: установить 7-й бит в `value` (сделать его = 1)

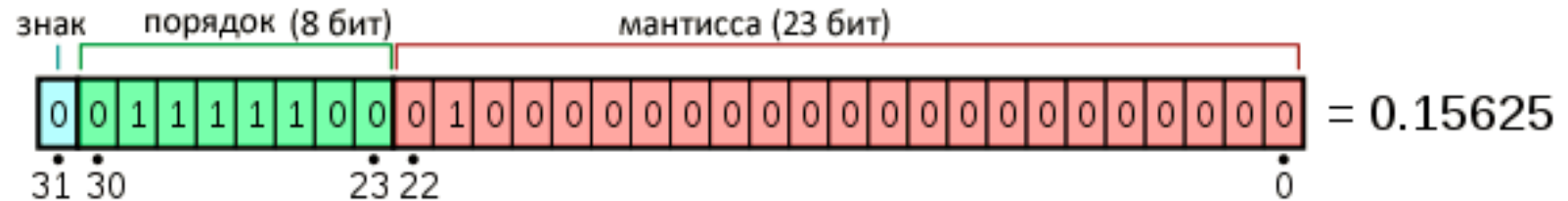
✓Решение: `value = value | (1 >> 7);`

Ввод и вывод в двоичном представлении:

- `std::vector<bool> // 0-й элемент вектора - самый старший бит. Чтобы записать число 0b00000001 нужно сделать 7й элемент вектора = 1`
- `std::bitset<8> // 8 - сколько бит вывести`

# Числа с плавающей запятой (floating-point numbers)

- Представлены в памяти нетривиально.
  - IEEE 754:  $x = M \cdot 2^E$ ,  $M$  и  $E$  целые.



*Рисунок взят из Википедии*

В этом примере:

- Знак  $s=0$  (положительное число)
- Порядок  $E=01111100_2-127_{10} = -3$
- Мантисса  $M = 1.01_2$  (первая единица не явная)
- В результате наше число  $F = 1.01_2 \cdot 2^{-3} = 2^{-3} + 2^{-5} = 0,125 + 0,03125 = 0,15625$



# Сравнение чисел с плавающей запятой

- Проверка на равенство:

```
float x = 0.3333333f;
```

```
float y = 1.0f / 3.0f;
```

```
if (x == y) // false из-за ошибки округления
```

```
if (abs(x - y) < N * EPS) // Корректно; но что такое N и EPS?
```

- EPS — «машинное эпсилон»,  
1.0f + EPS == 1.0f из-за конечной точности.
  - FLT\_EPSILON ( $2^{-23}$ ), DBL\_EPSILON ( $2^{-52}$ ) в <float>.
  - См. курс вычислительной математики (BM-2).
- N зависит от способа вычисления x и y,  
но на практике выбирают, например, N = 16.



# Числа с фиксированной запятой (fixed-point numbers)

- Дробные величины представляют целыми числами (пример: не 1 р. 50 к., а 150 к.).
- Нет потерь точности (у всех чисел она равна).
- Высокая производительность.
- Ограничен диапазон (в т. ч. снизу).
- Пример:
  - `using Money = uint16_t; // Деньги в копейках.`
  - `Money price = 20050; // 200 р. 50 к.`
  - Диапазон: `{0, 1 к., ..., 655 р. 35 к.}`

# Строки C (C-style strings)

Строка C — массив символов, завершающийся нулевым символом `'\0'`.

```
char greeting[] = "Hello!";
```

- Размер определится автоматически (работает для любых встроенных массивов).
- Длина строки — 6 символов.
- **sizeof**(greeting) == 7
- `// char greeting[7] { 'H', 'e', 'l', 'l', 'o', '!', '\0' };`

```
const char* farewell = "Goodbye!";
```

- **sizeof**(farewell) == 4 // размер указателя
- Длина строки — 8 символов, где-то в памяти их 9.

# Обработка строк C

```
size_t get_string_length ( const char* symbols )
```

```
{
```

```
    size_t length = 0;
```

```
    while ( *symbols ) {
```

```
        ++length;
```

```
        ++symbols;
```

```
    }
```

```
    return length;
```

```
}
```

Разыменование дает символ,  
на который указывает `symbols`.  
Если это `'\0'`, условие ложно.

Смещение указателя  
к адресу очередного символа.

× Если `symbols == nullptr`, нельзя делать `*symbols`.

# Копирование строк C

```
void copy_string(char* to, const char* from)
```

```
{  
    while (*from) {  
        *to = *from;  
        ++to;  
        ++from;  
    }  
    *to = *from;  
}
```

1) Пока есть символ для копирования,  
2) копировать его  
3) и перейти к следующей ячейке для копии,  
4) а также к следующему исходному символу.  
5) Скопировать нулевой символ.

**Предполагается**, что массив, на который указывает `to`, достаточно велик, чтобы вместить символы из `from`.

**Проверить** это в `copy_string()` **нельзя**.

# Работа со строками

## Класс `std::string`

```
string name, message;  
const string greeting = "Hello";  
getline ( cin, name );
```

```
message = greeting;  
message += ", " + name + "!";
```

```
cout << message << '\n';
```

## Строки C (`<cstring>`) <sup>48</sup>

```
char name [ 32 ], message [ 32 ];  
const char* greeting = "Hello";  
fgets ( name, sizeof(name), stdin );  
  
// gets() небезопасна! - не позволяет  
// установить ограничение на количество  
// считываемых символов, поэтому нужно быть  
// осторожными с размером массива, чтобы  
// избежать переполнения буфера.  
strcpy ( message, greeting );  
strcat ( message, ", " );  
strcat ( message, name );  
strcat ( message, "!" );  
puts ( message ); // cout << ...
```

# Литература к лекции

- *Programming Principles and Practices Using C++:*
  - глава 25 — тема лекции;
  - раздел 27.5 — строки C;
  - аналогичная [презентация](#) (скорее, наоборот :-).
- *C++ Primer:*
  - разделы 3.5 и 3.6 — подробно о массивах.
- *Сайт «C++ Reference»:*
  - функции для работы с памятью и строками;
  - ограничения типов с плавающей запятой;
  - описание `std::array`, `std::vector <bool>`, `std::bitset`.
- [Статья](#) о плавающей запятой.