C/C++: Лекция 2

Воробьев Д.В

11.09.2020

Память

Cell

Факты:

- Минимальная единица памяти, на которую можно сослаться
- Обладает адресом (address)
- Состоит из битов
- За историю разработки компьютеров размеры были разными
- На текущий момент размер обычно 8 бит

Address

Факты:

- Натуральное число для кодирования cell
- Комбинация слов "разрядность системы" = "степень 2-ки, позволяющая получить необходимое число натуральных чисел"

Byte

Факты:

- Для 8 битных cell введен термин byte

Соответственно:

- 1. На "32 разрядной системе" кодируется 2^{32} cells
- 2. 1 cell = 1 byte
- 3. Из п.1 и п.2 2^{32} byte. $\Rightarrow$ 4GB.

Word

Факты:

- Последовательность байт
- Для ”32 разрядной системы” по определению взято 4 байта
- Для ”64 разрядной системы” по определенеию взято 8 байт

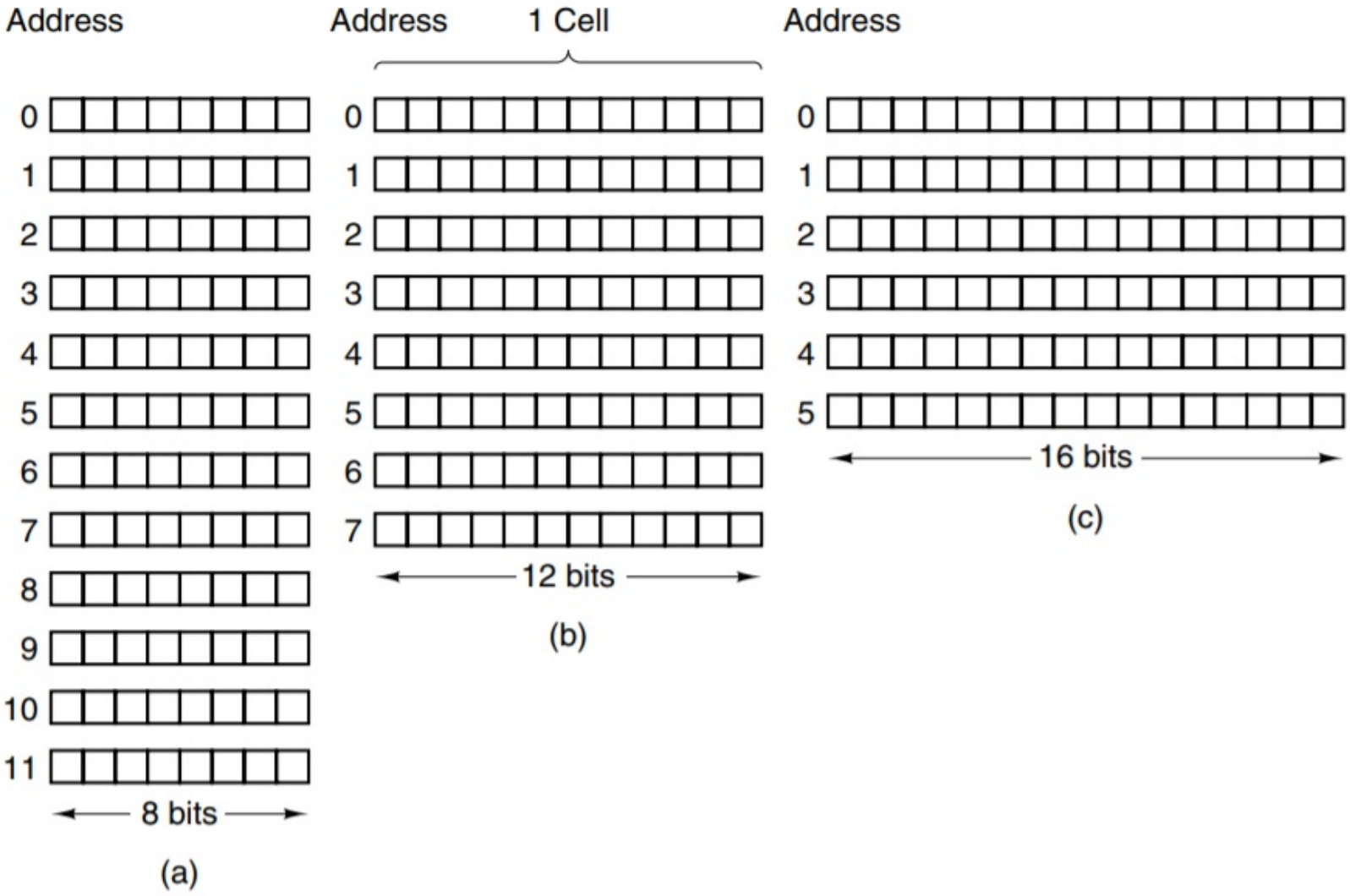


Figure: Варианты размеров cell

Computer	Bits/cell
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

Figure: Размеры cell в различных машинах

Endian

Предупреждение

По причине ”1 address соответствует 1 byte” будем взимозаменять address и byte.

Endian

Проблема

Дали число $6_{10} = 0000\ 0000\ 0000\ 0110_2$. Непонятно какому байту (номеру) в СЛОВЕ какой байт в МОДЕЛИ сопоставить.

Способы:

- BigEndian: БОЛЬШИЙ байт в модели в МЕНЬШИЙ байт в слове. (Big т.к. в такой формулировке слово ОКАНЧИВАЕТСЯ на БОЛЬШИЙ байт МОДЕЛИ)
- LittleEndian: БОЛЬШИЙ байт в модели БОЛЬШИЙ байт в слове (Little т.к. в такой формулировке слово ОКАНЧИНВАЕТСЯ на МЕНЬШИЙ байт МОДЕЛИ)

Endian

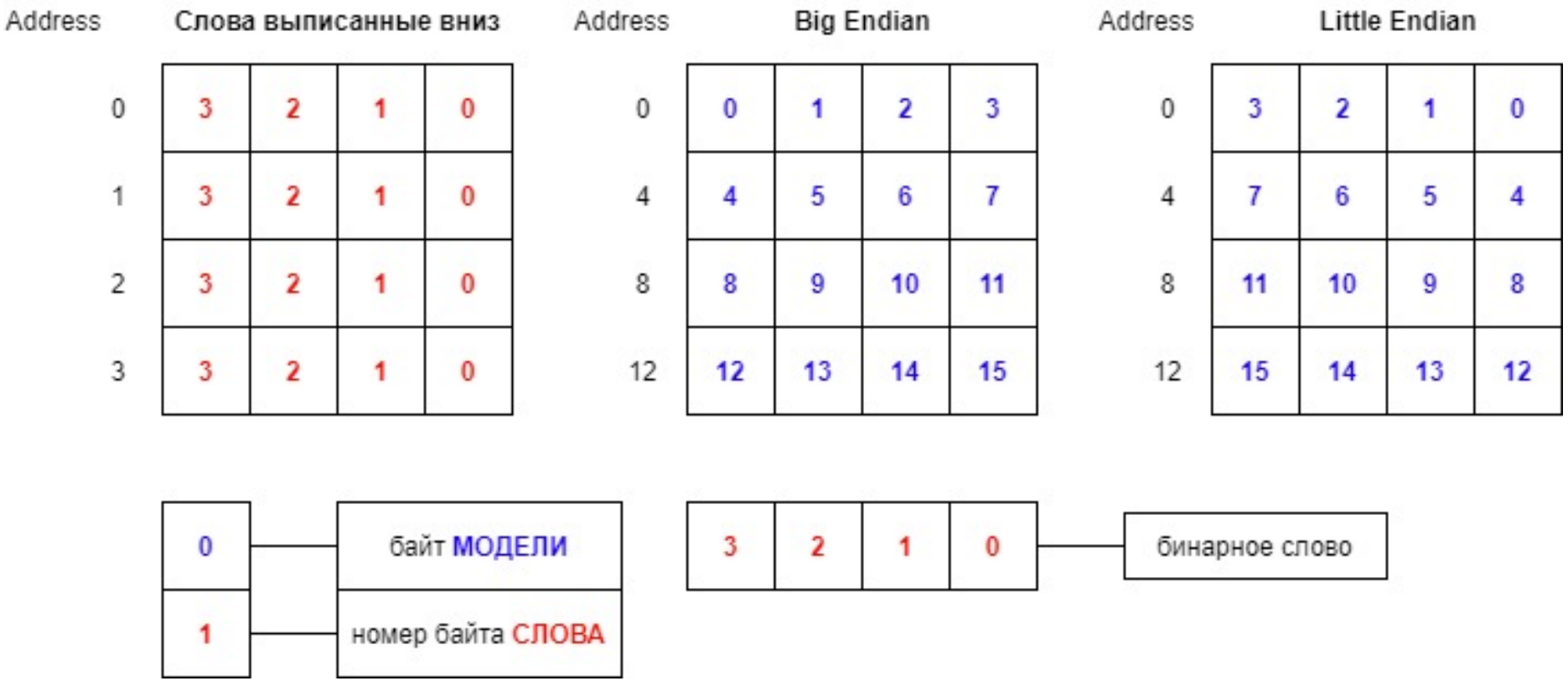


Figure: Биекция байтов МОДЕЛИ на байты СЛОВ

Операторы

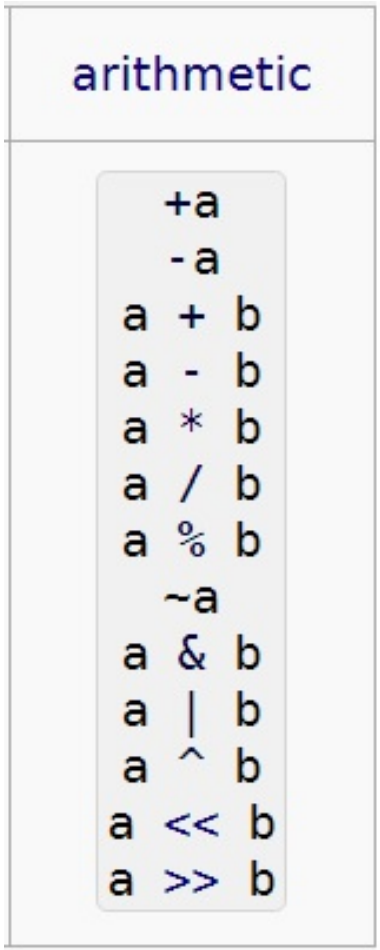
Precedence

Порядок вызова операторов.

Associativity

Свойство оператора, определяющее порядок расстановки скобок.

Арифметические



Арифметические

Правоассоциативные

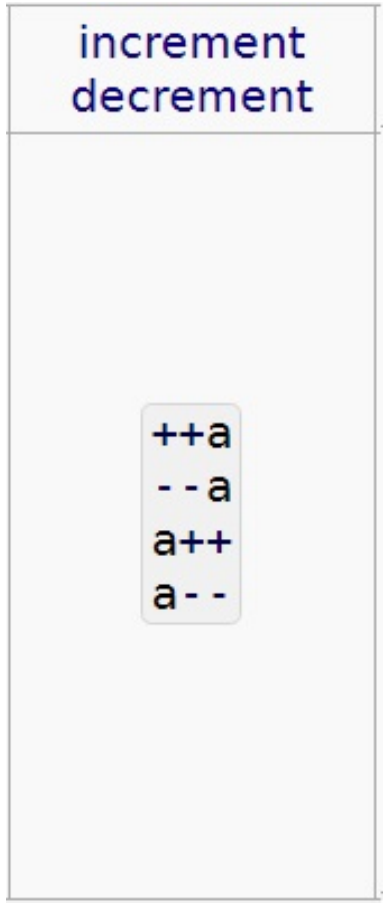
\+ (унарный) - (унарный)

Левоассоциативные

\+ (бинарный) - (бинарный) * / % | ^ « »

```
#include <iostream>

int main() {
    // ((7 - 7) - 7) = -7
    std::cout << 7 - 7 - 7;

    // ((24 / 4) / 2) = 3
    std::cout << 24 / 4 / 2;
    return 0;
}
```

Икремент / декримент



Икремент / декримент

Префиксный

1. увеличить значение
2. вернуть значение

```
#include <iostream>

int main() {
    int x = 10;
    // 11
    std::cout << ++x;
    // 11
    std::cout << x;
}
```

Икремент / декримент

Постфиксный

1. сохранить значение
2. увеличить значение
3. вернуть сохраненное

```cpp
#include <iostream>

int main() {
    int x = 10;
    // 10
    std::cout << x++;
    // 11
    std::cout << x;
}
```

Скомбиниурем изученные операторы

```
#include <iostream>

int main() {
    int x = 10;
    std::cout << ---x;
    return 0;
}
```

Видим - - и - с одинаковым приоритетом и правоассоциативные

3	`++a --a`
	`+a -a`
	`! ~`
	`(type)`
	`*a`
	`&a`
	`sizeof`
	`co_await`
	`new new[]`
	`delete delete[]`
		Right-to-left

Получаем

```
#include <iostream>

int main() {
    int x = 10;
    std::cout << --(-x);
    return 0;
}
```

Да, тут ошибка. Но это демонстрирует, что порядок действий правильный. Пример про associativity, а не value category.

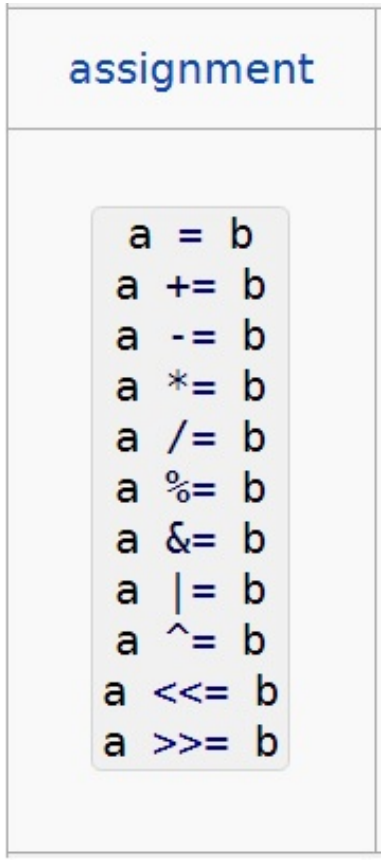
Причина ошибки

- 1. префиксный – принимает lvalue expression
- 2. у нас у это -x –
- 3. -x это prvalue expression
- 4. не состыковочки

```cpp
#include <iostream>

int main() {
    int x = 10;
    // тут все ок выведет 9
    std::cout << -(--x);
    return 0;
}
```

```cpp
#include <iostream>

int main() {
    int x = 10;
    // доказывает
    // что скобки так -(x--)
    // т.к. если так (-x)--
    // то была бы ошибка
    // но ее нет
    std::cout << -x--;
    return 0;
}
```

Присваивание

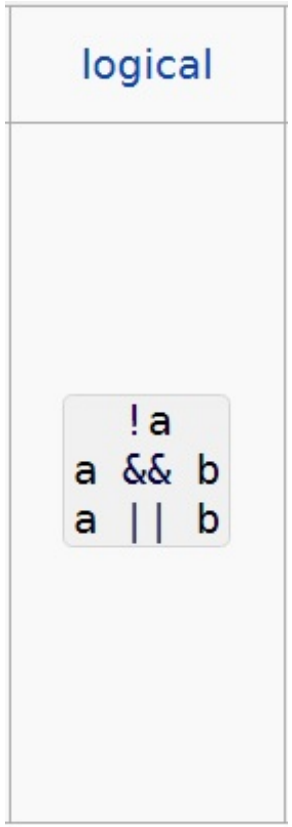


Присваивание

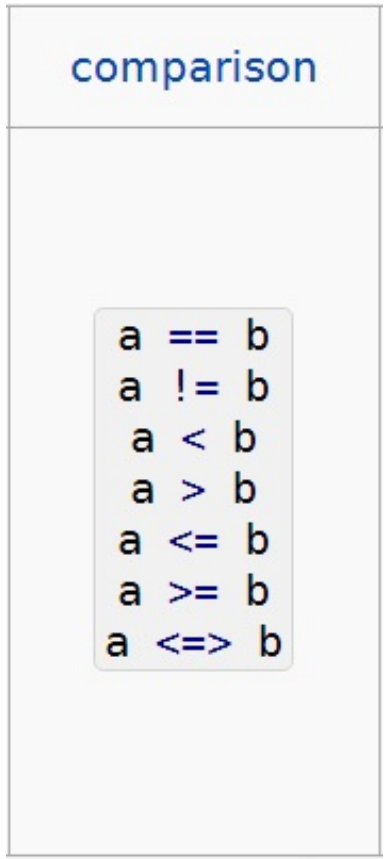
```cpp
#include <iostream>

int main() {
    int x = 10;
    int y = 6;

    x &= y;
    // 2
    std::cout << x;
    return 0;
}
```

```cpp
#include <iostream>

int main() {
    int x = 10;
    int y = 6;

    x = x & y;
    // 2
    std::cout << x;
    return 0;
}
```

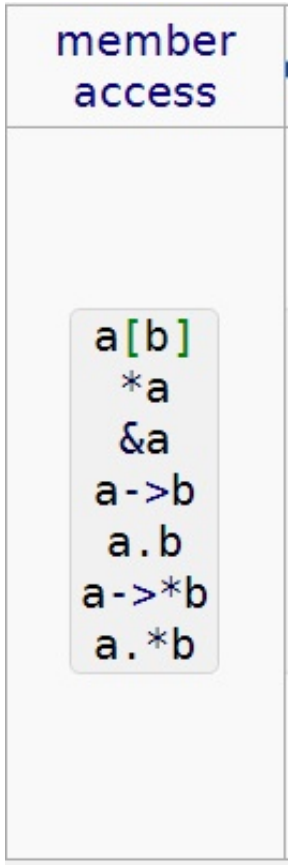
Логические



Сравнения



Доступа



Доступа

```cpp
#include <iostream>

struct A {
    int x = 10;
};

int main() {
    A* p = new A;
    std::cout << p->x;
    std::cout << (*p).x;

    return 0;
}
```

```cpp
#include <iostream>

int main() {
    int* p = new (5);
    // 5
    std::cout << *p;
    int x = 10;
    p = &x;
    // 10
    std::cout << p;

    return 0;
}
```

.* use case

При переименовании member

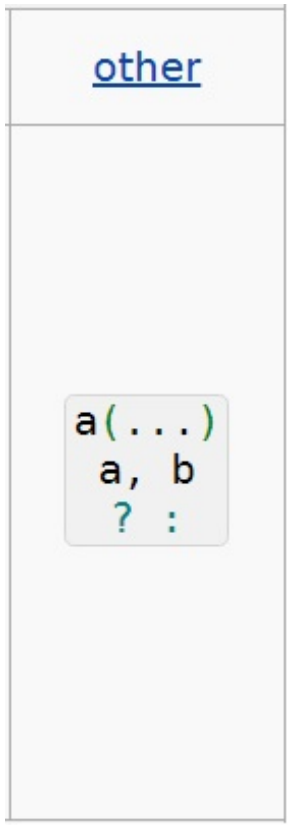
2 изменения

```cpp
#include <iostream>

struct S {
    int member = 10;
};

int main() {
    int S::* ptr = &S::member;
    S a;
    std::cout << a.*ptr;
    S b;
    std::cout << b.*ptr;
    S c;
    std::cout << c.*ptr;
    return 0;
}
```

4 изменения

```cpp
#include <iostream>

struct S {
    int member = 10;
};

int main() {
    S a;
    std::cout << a.member;
    S b;
    std::cout << b.member;
    S c;
    std::cout << c.member;

    return 0;
}
```

Другие



Тернарный

```
#include <iostream>

int main() {
    int x = 0;
    int y = 1;

    std::cout << a > b ? a : b;
    return 0;
}
```

Запятая

```cpp
#include <iostream>

int main() {
    int n = 1;
    int m = (++n, std::cout << "n = " << n << '\n', ++n, 2*n);
    std::cout << "m = " << (++m, m) << '\n';

    return 0;
}
```

sizeof, alignof

- sizeof - возвращает размер объекта
- alignof - возвращает число байт требуемое для выравнивания

```
#include <iostream>

struct C {
    char x;
    int y;
};
int main() {
    // 1
    std::cout << alignof(char);
    // 4
    std::cout << alignof(int);
    // выравнивание по int
    std::cout << sizeof(C);
    return 0;
}
```

lvalue и rvalue

Определение

lvalue - expression, которому можно сделать присвоение

rvalue - не lvalue expression

Предупреждение

Это грубое определение. Более аккуратно во 2-ой части курса.

Перегрузка функций

Перегрузка функции

Определение более 2-ух функций в одинаковом scope с разным списком параметров.

Замечание

Определение не полное для методов структуры / класса требуется уточнение.

Перегрузка функций

Можно

```
#include <iostream>

void func(double a) {}
void func(int a) {}

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

Нельзя

```
#include <iostream>

void func(int a) {}
int func(int a) {}

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

Функции с аргументами по умолчанию

Указываются последними в списки параметров

Можно

```
#include <iostream>

void func(int a, int b = 0) {}

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

Нельзя

```
#include <iostream>

void func(int b = 0, int a) {}

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

Явное приведение типов

Проблема

Слева дали double* указатель на float. При чтении будет считываться 8 байт. Памяти выделяли под 4 байта

C-cast

```cpp
#include <iostream>

int main() {
    float x = 3.1;
    // UB
    double* y = (double*) &x;
    return 0;
}
```

static_cast

```cpp
#include <iostream>

int main() {
    float x = 3.1;
    // CE
    double* y = static_cast<double*>(&x);
    return 0;
}
```

Вывод

Используйте static_cast. static_cast проверит совместимость типов

Управляющие конструкции

if

C++

```
int main() {
    int x = 10;
    if (x) {
        int y = 20;
    }
    int z = 10;
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
    // сравниваем x из if с 0
    // пишем результат в регистр
    cmp     dword ptr [rbp - 8], 0
    // jump по метке по результату
    je      .LBB0_2
    mov     dword ptr [rbp - 12], 20
.LBB0_2:
    xor     eax, eax
    mov     dword ptr [rbp - 16], 10
    pop     rbp
    ret
```

if, else

C++

```
int main() {
    int x = 10;
    if (x) {
        int y = 20;
    } else {
        int y = 5;
    }
    int z = 10;
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
    cmp     dword ptr [rbp - 8], 0
    // метка if
    je      .LBB0_2
    mov     dword ptr [rbp - 12], 20
    // метка else
    jmp     .LBB0_3
.LBB0_2:
    mov     dword ptr [rbp - 16], 5
.LBB0_3:
    xor     eax, eax
    mov     dword ptr [rbp - 20], 10
    pop     rbp
    ret
```

Dangling else

```
#include <iostream>

int main() {
    int x = 0;
    if (1)
        if (1)
            x = 1;
    else
        x = 2;
    return 0;
}
```

```
#include <iostream>

int main() {
    int x = 0;
    if (0)
        if (0)
            x = 1;
    else
        x = 2;
    return 0;
}
```

```
#include <iostream>

int main() {
    int x = 0;
    if (1)
        if (0)
            x = 1;
    else
        x = 2;
    return 0;
}
```

Вывод

Ставьте скобки и пишите всегда явно

while

C++

```
int main() {
    int x = 0;
    while(x < 1) {x++;}
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
.LBB0_1:
    cmp     dword ptr [rbp - 8], 1
    jge     .LBB0_3
    mov     eax, dword ptr [rbp - 8]
    add     eax, 1
    mov     dword ptr [rbp - 8], eax
    jmp     .LBB0_1
.LBB0_3:
    xor     eax, eax
    pop     rbp
    ret
```

do-while

C++

```
int main() {
    int j = 0;
    do {
        j++;
    } while (j < 2);
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
.LBB0_1:
    mov     eax, dword ptr [rbp - 8]
    add     eax, 1
    mov     dword ptr [rbp - 8], eax
    cmp     dword ptr [rbp - 8], 2
    jl      .LBB0_1
    xor     eax, eax
    pop     rbp
    ret
```

for

C++

```
int main() {
    for(std::size_t x = 0; x < 1; 
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
.LBB0_1:
    cmp     dword ptr [rbp - 8], 1
    jge     .LBB0_4
    jmp     .LBB0_3
.LBB0_3:
    mov     eax, dword ptr [rbp - 8]
    add     eax, 1
    mov     dword ptr [rbp - 8], eax
    jmp     .LBB0_1
.LBB0_4:
    xor     eax, eax
    pop     rbp
    ret
```

switch

C++

```
int main() {
    int x = 0;
    switch(x) {
        case 0 : {
            int y = 1;
            break;
        }
        default: {
            int y = 2;
        }
    }
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
    mov     eax, dword ptr [rbp - 8]
    test    eax, eax
    jne     .LBB0_2
    jmp     .LBB0_1
.LBB0_1:
    mov     dword ptr [rbp - 12], 1
    // это jump от break
    jmp     .LBB0_3
.LBB0_2:
    mov     dword ptr [rbp - 16], 2
.LBB0_3:
    xor     eax, eax
    pop     rbp
```

switch

C++

```
int main() {
    int x = 0;
    switch(x) {
        case 0 : {
            int y = 1;
        }
        default: {
            int y = 2;
        }
    }
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
    mov     eax, dword ptr [rbp - 8]
    test    eax, eax
    jne     .LBB0_2
    jmp     .LBB0_1
.LBB0_1:
    mov     dword ptr [rbp - 12], 1
// без break пойдем в след. секцию
.LBB0_2:
    mov     dword ptr [rbp - 16], 2
    xor     eax, eax
    pop     rbp
    ret
```

break

break

jump в метку конца цикла

C++

```
int main() {
    int x = 10;
    while( x < 10) {
        break;
    }
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
    cmp     dword ptr [rbp - 8], 10
    jge     .LBB0_3
    // это jump от break
    jmp     .LBB0_3
.LBB0_3:
    xor     eax, eax
    pop     rbp
    ret
```

continue

continue

jump в метку начала цикла

C++

```
int main() {
    int x = 10;
    while( x < 10) {
        continue;
    }
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
.LBB0_1:
    cmp     dword ptr [rbp - 8], 10
    jge     .LBB0_3
    // это jump continue
    jmp     .LBB0_1
.LBB0_3:
    xor     eax, eax
    pop     rbp
    ret
```

Return

C++

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

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    xor     eax, eax
    mov     dword ptr [rbp - 4], 0
    // return 0
    pop     rbp
    ret
```