

С/С++: Лекция 2

Воробьев Д.В

11.09.2020

Операторы

Precedence

Порядок вызова операторов.

Associativity

Свойство оператора, определяющее порядок расстановки скобок.

Арифметические

arithmetic

`+a`

`-a`

`a + b`

`a - b`

`a * b`

`a / b`

`a % b`

`~a`

`a & b`

`a | b`

`a ^ b`

`a << b`

`a >> b`

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

increment
decrement

```
++a  
--a  
a++  
a--
```

Икремент / декремент

Префиксный

1. увеличить значение
2. вернуть значение

```
#include <iostream>

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

Икремент / декремент

Постфиксный

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

```
#include <iostream>

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


Скомбинируем изученные операторы

```
#include <iostream>

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

Видим - - и - с одинаковым приоритетом и правоассоциативные

| | | |
|---|---|---------------|
| 3 | <code>++a --a</code> <code>+a -a</code> <code>! ~</code> <code>(type)</code> <code>*a</code> <code>&a</code> <code>sizeof</code> <code>co_await</code> <code>new new[]</code> <code>delete delete[]</code> | 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

```
#include <iostream>

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

```
#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

assignment

```
a = b  
a += b  
a -= b  
a *= b  
a /= b  
a %= b  
a &= b  
a |= b  
a ^= b  
a <<= b  
a >>= b
```

Присваивание

```
#include <iostream>

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

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

```
#include <iostream>

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

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

logical

```
! a
a && b
a || b
```


Сравнения

comparison

```
a == b  
a != b  
a < b  
a > b  
a <= b  
a >= b  
a <=> b
```

Доступа

member
access

```
a[b]  
*a  
&a  
a->b  
a.b  
a->*b  
a.*b
```

Доступа

```
#include <iostream>

struct A {
    int x = 10;
};

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

    return 0;
}
```

```
#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

```
#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

```
#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

other

```
a(...)  
a, b  
? :
```

Тернарный

```
#include <iostream>

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

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

Запятая

```
#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

```
#include <iostream>

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

static_cast

```
#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

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


if, else

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

while

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

do-while

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

for

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

switch

```
int main() {  
    int x = 0;  
    switch(x) {  
        case 0 : {  
            std::cout << 1;  
            break;  
        }  
        case 1 : {  
            std::cout << 2;  
            break;  
        }  
        default: {  
            std::cout << 3;  
        }  
    }  
    // 1  
    return 0;  
}
```

switch

```
int main() {  
    int x = 0;  
    switch(x) {  
        case 0 : {  
            std::cout << 1;  
        }  
        case 1 : {  
            std::cout << 2;  
            break;  
        }  
        default: {  
            std::cout << 3;  
        }  
    }  
    // 12  
    return 0;  
}
```

switch

```
int main() {  
    int x = 2;  
    switch(x) {  
        case 0 : {  
            std::cout << 1;  
        }  
        case 1 : {  
            std::cout << 2;  
            break;  
        }  
        default: {  
            std::cout << 3;  
        }  
    }  
    // 3  
    return 0;  
}
```

break

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


continue

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

Return

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