

Конструкторы копирования и прочее

Сабалевский Сергей, 2019



Оглавление

1. [Повторение](#)
2. [Предисловие](#)
3. [Конструктор копирования](#)
4. [Перегрузка оператора присваивания](#)
5. [Дополнительно: NULL и nullptr](#)
6. [Дополнительно: Делегирующие конструкторы](#)

Повторение

[Вернуться к оглавлению](#)



Конструктор:

1. Конструктор существует всегда
2. Конструкторы можно перегружать
3. Они упрощают нам создание объектов класса
4. Вызывается автоматически при создании объекта класса
5. Нельзя вызвать напрямую (как метод класса)

Деструктор:

1. Всегда единственный
2. Выполняется при уничтожении объекта (завершении работы программы или при выходе из зоны видимости)
3. Нельзя вызвать напрямую
4. Очищает использованную память

Конструкторы

```
class myclass {  
  
};
```

```
class myclass {  
    int x;  
public:  
    void set_x(int x);  
    int get_x();  
};
```

```
class myclass {  
    int x;  
public:  
    myclass() { }  
    myclass(int x);  
  
    void set_x(int x);  
    int get_x();  
};  
myclass::myclass(int x) { ... }
```

Деструктор

```
class myclass {  
    int *p;  
    int a;  
public:  
    myclass(int a);  
    ~myclass() {  
        delete p;  
    }  
};
```

```
myclass::myclass(int a) {  
    this->a = a;  
    //a = a; - ни к чему не  
    приведет  
    p = &a;  
}
```

Оператор this

```
myclass::myclass(int a) {  
    this->a = a;  
    //a = a; - ни к чему не  
    приведет  
    p = &a;  
    cout << "a = " << a << endl  
    << "p = " << p << endl;  
}
```

```
class Foo  
- Print();
```

Foo f;

this

Адрес f

Foo f1;

this

Адрес f1

```
f1.Print();
```

```
#define _CRT_SECURE_NO_WARNINGS

...

int main () {
    setlocale(LC_ALL, "Russian");

    ...

    return 0;
}
```

Предисловие

[Вернуться к оглавлению](#)

Почему код выполнится с ошибкой?

```
class strtype {
    char *p;
public:
    strtype(const char *s) {
        int l = strlen(s) + 1;
        p = new char[l];
        strcpy(p, s);
    }
    ~strtype() {
        cout << "delete " << p << endl;
        delete[] p;
    }
    char *get() { return p; }
};

void show(strtype x) {
    char *s;
    s = x.get();
    cout << s << endl;
}

int main() {
    strtype a("First"), b("Second");
    show(a);
    show(b);
    return 0;
}

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
```

Как заставить работать правильно?

```
class strtype {  
    char *p;  
public:  
    strtype(const char *s) {  
        int l = strlen(s) + 1;  
        p = new char[l];  
        strcpy(p, s);  
    }  
    ~strtype() {  
        cout << "delete " << p << endl;  
        delete[] p;  
    }  
    char *get() { return p; }  
};
```

```
void show(strtype x) {  
    char *s;  
    s = x.get();  
    cout << s << endl;  
}  
  
int main() {  
    strtype a("First"), b("Second");  
    show(a);  
    show(b);  
    return 0;  
}
```

```
#define _CRT_SECURE_NO_WARNINGS  
#include <iostream>  
#include <cstring>  
#include <cstdlib>  
using namespace std;
```

Конструктор копирования

[Вернуться к оглавлению](#)

**Давайте рассмотрим
способы передачи
параметров в функцию**

—

Пусть у нас есть такой класс

```
class myclass {  
    int x;  
public:  
    myclass(int n) : x(n) { cout << "-Create " << x << endl; }  
    ~myclass() { cout << "--Delete " << x << endl; }  
    int get_x() { return x; }  
    void set_x(int a) { x = a; }  
};
```

```
void func0(myclass);  
void func1(myclass &);  
void func2(myclass *);
```

Рассмотрим сами функции

```
void func0(myclass ob) {
    cout << " Local x: " << ob.get_x() << endl;
    ob.set_x(10);
    cout << " Local x: " << ob.get_x() << endl;
}

void func1(myclass &ob) {
    cout << " Local x: " << ob.get_x() << endl;
    ob.set_x(11);
    cout << " Local x: " << ob.get_x() << endl;
}

void func2(myclass *ob) {
    cout << " Local x: " << ob->get_x() << endl;
    ob->set_x(12);
    cout << " Local x: " << ob->get_x() << endl;
}
```

B main 0

```
myclass ob(0);  
cout << "&ob0: " << &ob << endl;  
cout << "in main(): " << ob.get_x() << endl;  
func0(ob);  
cout << "in main(): " << ob.get_x() << endl;
```

```
-Create 0  
&ob0: 00BCFAB4  
in main(): 0  
  Local x: 0  
    &ob: 00BCF984  
  Local x: 10  
--Delete 10  
in main(): 0
```


B main 1

```
myclass ob1(1);  
cout << "&ob1: " << &ob1 << endl;  
cout << "in main(): " << ob1.get_x() << endl;  
func1(ob1);  
cout << "in main(): " << ob1.get_x() << endl;
```

```
-Create 1  
&ob1: 00BCFAA8  
in main(): 1  
  Local x: 1  
    &ob: 00BCFAA8  
  Local x: 11  
in main(): 11
```

B main 2

```
myclass* ob2 = new myclass(2);
cout << "&ob2: " << &ob2 << " ob2: " << ob2
<< endl;
cout << "in main(): " << ob2->get_x() <<
endl;
func2(ob2);
cout << "in main(): " << ob2->get_x() <<
endl;
delete ob2;
// Мы можем таким образом вызвать деструктор
```

```
-Create 2
&ob2: 00BCFA9C ob2: 01085CB0
in main(): 2
Local x: 2
    &ob: 00BCF984 ob: 01085CB0
Local x: 12
in main(): 12
--Delete 12
```

Есть вопросы?

[Вернуться к оглавлению](#)

Вернемся к конструкторам копирования

Конструктор копирования

```
class strtype {  
    char *p;  
public:  
    strtype(char *s);           // конструктор  
    strtype(const strtype &other);  
    // конструктор копий  
    ~strtype() {delete [] p;} // деструктор  
    char *get() {return p;}  
};
```

//Общая форма конструктора копирования имеет вид:

```
имя_класса (const имя_класса &o) {  
    // тело конструктора  
}
```

```
class myclass {...}  
myclass func() {...}  
myclass B;  
void func1(myclass ob) {...}  
// Конструктор копий выполняется в следующих  
случаях  
    B = func();  
    myclass A = B;  
    func1(A);
```

Продолжение

```
strtype::strtype(const char *s) {  
    int r;  
    r = strlen(s) + 1;  
    p = new char[r];  
    strcpy(p, s);  
}
```

```
strtype::strtype(const strtype &other) {  
    int r;  
    r = strlen(o.p) + 1;  
    p = new char[r];  
    strcpy(p, o.p);  
    // p[strlen(o.p) - 1] = '0';  
}
```

```
void show(strtype x) {  
    char *s;  
    s = x.get();  
    cout << s << "\n";  
}
```


```
int main() {  
    strtype a("First"), b("Second");  
    show(a);  
    show(b);  
    return 0;  
}
```

Закрепим написание программы


Наш класс

```
class myclass {  
public:  
    int size;  
    int* data;  
    myclass(int size);  
  
    ~myclass();  
};
```

```
myclass::~myclass() {  
    cout << " Destructor " << this << endl;  
    delete[] data;  
}  
  
myclass func(int size);
```

```
myclass::myclass(int size) {  
    this->size = size;  
    this->data = new int[size];  
}
```



```
myclass::myclass(int size) {  
    this->size = size;  
    this->data = new int[size];  
}
```

```
myclass::myclass(int size) {
    this->size = size;
    this->data = new int[size];

    for (int i = 0; i < size; i++) {
        data[i] = i;
    }
    cout << " Constructor " << this << endl;
}
```

```
myclass func(int size) {
    cout << "myclass temp(size);" << endl;
    myclass temp(size + 1);
    cout << "return temp;" << endl;
    return temp;
}
```

```
myclass::myclass(const myclass &other) {
    this->size = other.size;
    this->data = new int[other.size];

    for (int i = 0; i < other.size; i++) {
        this->data[i] = other.data[i];
    }
    cout << " Constructor copy " << this << endl;
}
```

```
int main() {  
    myclass a(5);  
  
    myclass b(a);  
  
    myclass c = b;  
  
    myclass d = func(5);  
    cout << "End" << endl;  
    system("pause");  
    return 0;  
}
```

For Microsoft vc++
Constructor 008FFA04

Constructor copy 008FF9F4

Constructor copy 008FF9E4

Constructor 008FF8C8
Constructor copy 008FF9D4
Destructor 008FF8C8
End
Destructor 008FF9D4
Destructor 008FF9E4
Destructor 008FF9F4
Destructor 008FFA04

Есть вопросы?

[Вернуться к оглавлению](#)

А что будет, если сделать так?

```
int main() {  
    cout << "myclass a(5);" << endl;  
    myclass a(5);  
    cout << "myclass b(5);" << endl;  
    myclass b(5);  
    cout << "a = b;" << endl;  
    a = b;  
    system("pause");  
    return 0;  
}
```

Constructor 0073F7A0

Constructor 0073F790

Для продолжения нажмите любую клавишу . . .

Destructor 0073F790

Destructor 0073F7A0

... и вылетит с ошибкой

Перегрузка оператора присваивания

[Вернуться к оглавлению](#)

Оператор =

Все операторы в C++ можно считать синтаксическим сахаром, скрывающем конкретные функции/

Но поговорим конкретно про присваивание.

А он вообще что-нибудь возвращает?

```
int a = 4;  
int b;  
b = a;
```


Его можно переопределить

```
void operator = (const myclass &other) {  
    cout << "operator = " << this << endl;  
}
```

```
myclass& operator = (const myclass &other) {  
    cout << "operator = " << this << endl;  
    return *this;  
}
```

Что на счет множественного присваивания?

```
int a, b, c;  
c = 9;  
a = b = c;
```

Самая правильная реализация

```
myclass& operator = (const myclass &other) {  
    cout << "operator " << this << " = " <<  
    &other << endl;  
  
    this->size = other.size;  
    if (this->data != nullptr) {  
        delete[] this->data;  
    }  
    this->data = new int[other.size];  
  
    for (int i = 0; i < other.size; i++) {  
        this->data[i] = other.data[i];  
    }  
    return *this;  
}
```

```
myclass a(5);  
    Constructor 0x61fe30  
c = func(5);  
    myclass temp(size);  
    Constructor 0x61fe20  
    return temp;  
operator 0x61fe30 = 0x61fe20  
End  
    Destructor 0x61fe20  
    Destructor 0x61fe30
```

Есть вопросы?

[Вернуться к оглавлению](#)

Дополнительно: NULL и nullptr

[Вернуться к оглавлению](#)

NULL = 0

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

10

```
int main() {
    int *pa = new int;
    *pa = 10;
    cout << *pa << endl;
    pa = NULL;
    delete pa;
    return 0;
}
```

В чем проблема?

```
int  *pa = new int;  
*pa = 10;  
cout << *pa << endl;  
delete pa;  
pa = 0; // == NULL
```

nullptr появился в C++11

```
int  *pa = new int;
*pa = 10;
cout << *pa << endl;
/* BANNED
pa = nullptr;
delete pa;
*/
if (pa != nullptr) {
    delete pa;
    pa = nullptr;
}
```

Дополнительно: Делегирующие конструкторы

[Вернуться к оглавлению](#)


```
class human {
    int age, weight, name;
public:
    human(int name) {
        this->name = name;
        this->age = 0;
        this->weight = 0;
    }
    human(int name, int age) {
        this->name = name;
        this->age = age;
        this->weight = 0;
    }
    human(int name, int age, int weight) {
        this->name = name;
        this->age = age;
        this->weight = weight;
    }
};
```

```
class human {
    int age, weight, name;
public:
    human(int name) {
        this->name = name;
        this->age = 0;
        this->weight = 0;
    }
    human(int name, int age) :human(name) {
        this->age = age;
    }
    human(int name, int age, int weight) :human(name, age)
    {
        this->weight = weight;
    }
};
```

Спасибо за внимание

[Вернуться к оглавлению](#)