

ОБЪЕКТНО- ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ





STACK / C

//stack.h

```
typedef struct{ /*...*/ } Stack;
```

Constructor & destructor

```
Stack* stack_create();
```

```
void stack_destroy(Stack* const stack);
```

```
void stack_clear(Stack* const stack);
```

```
void stack_push(Stack* const stack, int node);
```

Methods

```
void stack_pop(Stack* const stack);
```

```
int* stack_top(const Stack* const stack);
```

```
size_t stack_count(const Stack* const stack);
```

Constant methods

STACK / C++

```
//stack.h
```

```
/*typedef не нужен*/
```

```
class Stack{
```

```
    size_t size; ← Member variables
```

```
    /*...*/
```

```
public:
```

```
    Stack(); ← Constructor
```

```
    ~Stack(); ← Destructor
```

```
    void clear();
```

```
    void push(int node); ← Methods (member functions)
```

```
    void pop();
```

```
    int* top() const;
```

```
    size_t count() const; ← Constant methods  
                           (const member functions)
```

```
};
```

STACK / C++

```
//stack.cpp
#include "stack.h"

Stack::Stack(){
    size = 0;
    /*initialization of
       member variables*/
}

Stack::~Stack(){
    /*release of resources*/
}

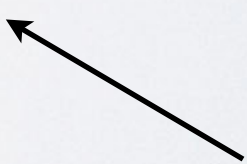
void Stack::clear(){
    /*delete elements*/
    size = 0;
}
```

```
//stack.cpp
void Stack::push(int node){
    /*add element*/
    ++size;
}

void Stack::pop(){
    /*delete element*/
    --size;
}

size_t Stack::count() const{
    //size++;    <= compile error
    return size;
}
```

cannot modify member variables
in const methods



РАБОТА С КЛАССАМИ

```
//main.cpp
```

```
#include <iostream>
```

```
#include "stack.h"
```

```
int main(){  
    {
```

Constructor call

```
        Stack stack;  
        stack.push(1);
```

```
        printf("Top element = %d;\n",  
               *stack.top());
```

```
        printf("Stack size = %d;\n",  
               stack.count());
```

```
    }
```

Implicit destructor call

```
    return 0;
```

```
}
```

Не вызывайте
сами деструктор!

РАБОТА С КЛАССАМИ

```
//main.cpp
```

```
#include <iostream>
```

```
#include "stack.h"
```

```
int main(){  
    {
```

```
        const Stack stack;
```

```
        stack.push(1);
```

← Constant object

← Compile error!
Method is not constant.

```
        printf("Stack size = %d;\n",  
               stack.count());
```

← Ok! Method is constant.

```
        const Stack* stackPtr = &stack;
```

```
        stackPtr->clear();
```

← Compile error!
Method is not constant.

```
    }  
  
    return 0;
```

```
}
```

```
//stack.h
```

```
class Stack{  
    size_t size;  
    /*...*/
```

```
public:
```

```
    Stack();
```

```
    ~Stack();
```

```
    void clear();
```

```
    void push(int node);
```

```
    void pop();
```

```
    int* top() const;
```

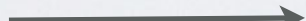
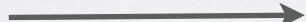
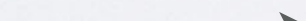
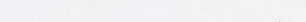

```
    size_t count() const;
```


```
};
```

РАБОТА С КЛАССАМИ

1. Константные методы могут вызывать любые объекты.
2. Неконстантные методы могут вызывать только неконстантные объекты.

ЧТО СКРЫВАЕТ КОМПИЛЯТОР?

```
int main(){  
    {  
        Stack stack;            Stack stack;  
        stack.push(1);           stack_ctor(&stack);  
        stack.top();             stack_push(&stack, 1);  
        stack.count();           stack_top(&stack);  
    }                             stack_count(&stack);  
    return 0;  
}
```

 stack_dtor(&stack);

Имена функций выдуманные. Компилятор
кодирует названия функций по-другому.

ЧТО СКРЫВАЕТ КОМПИЛЯТОР?

```
//stack.cpp
#include "stack.h"

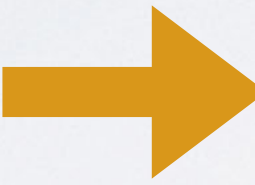
Stack::Stack(){...}

Stack::~~Stack(){...}

void Stack::clear(){...}

void Stack::push(int i){...}

size_t Stack::count() const{...}
```



```
//stack.cpp
#include "stack.h"

void stack_ctor(Stack*const this){...}

void stack_dtor(Stack*const this){...}

void stack_clear(Stack*const this){...}

void stack_push(Stack*const this, int i){...}

size_t stack_count(const Stack*const this){...}
```

Методы класса не хранятся
внутри объектов!
Это всего лишь функции.



cannot modify member variables
in const methods

KEYWORD THIS

```
//complex.h
class Complex{
    double Re, Im;
    /*...*/
public:
    Complex(double Re, double Im);
    ...
};

//complex.cpp
Complex::Complex(double Re, double Im){
    //Re = Re; ???
    //Im = Im; ??? WTF
}
```

KEYWORD THIS

```
//complex.h
class Complex{
    double Re, Im;
    /*...*/
public:
    Complex(double Re, double Im);
    ...
};
```

compiler will give precedence
to the local variables

```
//complex.cpp
Complex::Complex(double Re, double Im){
    this->Re = Re;
    this->Im = Im;
}
```

`void complex_ctor(Complex*const this, double Re, double Im)`

NEW / DELETE VS MALLOC / FREE

```
int* intPtr = new int;  
// int* intPtr = (int*)malloc(sizeof(int));
```

```
int* arrint = new int[1000];  
// int* arrint = (int*)malloc(1000*sizeof(int));
```

```
delete intPtr;  
// free(intPtr);
```

```
delete[] arrint;  
// free(arrint);
```

NEW / DELETE VS MALLOC / FREE

Foo* fooPtr = new Foo(); ← Constructor call

// Foo* fooPtr = (Foo*)malloc(sizeof(Foo));

// ... и вызвать foo_ctor()

Constructor without
args is called for each
object.

Foo* fooArr = new Foo[1000];

// Foo* fooArr = (Foo*)malloc(1000*sizeof(Foo));

// ... и вызвать foo_ctor() 1000 раз для каждого элемента

delete fooPtr; ← Destructeur call

// вызвать foo_dtor

// free(fooPtr);

Destructeur is called for
each object.

delete[] fooArr;

// ... вызвать foo_dtor 1000 раз для каждого элемента

// ... затем вызвать free(fooArr);



Попробуем реализовать **new** / **delete** на C?

NEW / DELETE HA C

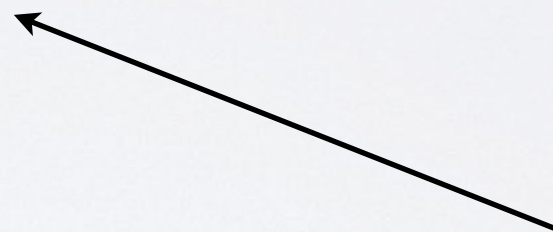
//new.h

```
void* new(const void* class, ...);  
void delete(void* self);
```

Pointer to class info



Variable number of
arguments for the
constructor



NEW / DELETE HA C

//new.h

```
typedef struct{  
    size_t size;  
    void *(*ctor)(void* self, va_list* app);  
    void *(*dtor)(void* self);  
}Class;
```

← Class info

```
void* new(const void* class, ...);  
void delete(void* self);
```

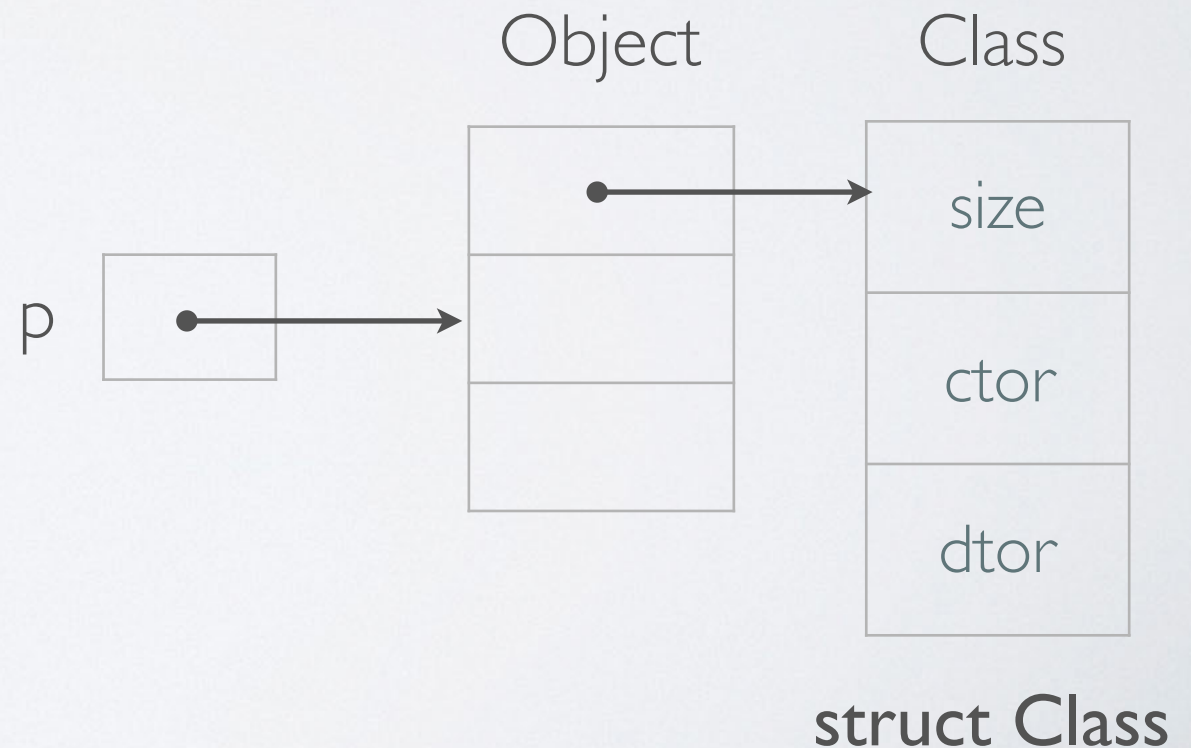
NEW / DELETE HA C

//point.h

```
struct Point{  
    const void* class;  
    int x, y;  
};
```

```
extern const void* Point;
```

Pointer to class info
about "struct Point"



NEW / DELETE HA C

```
//point.c
```

```
...
```

```
#include "point.h"
```

```
static void* point_ctor(void* _self,  
                        va_list* app)
```

```
{  
    struct Point *self = _self;  
    self->x = va_arg(*app, int);  
    self->y = va_arg(*app, int);  
    return self;  
}
```

```
static const Class _Point = {  
    sizeof(struct Point),    // size  
    point_ctor,             // ctor  
    NULL                    // dtor  
};
```

```
const void* Point = &_amp;_Point;
```

```
//new.h
```

```
typedef struct{  
    size_t size;  
    void *(*ctor)(void* self,  
                  va_list* app);  
    void *(*dtor)(void* self);  
}Class;
```

```
//point.h
```

```
struct Point{  
    const void* class;  
    int x, y;  
};  
  
extern const void* Point;
```

NEW / DELETE HA C

//new.c

```
void* new(const void* _class, ...)
{
    const Class* class = _class;
    void* p = calloc(1, class->size);
    assert(p);
    *(const Class**)p = class;

    if (class->ctor) {
        va_list ap;
        va_start(ap, _class);
        p = class->ctor(p, &ap);
        va_end(ap);
    }

    return p;
}
```

NEW / DELETE HA C

```
//new.c
```

```
...
```

```
void delete(void* self)
```

```
{
```

```
    const Class **cp = self;
```

```
    if (self && *cp && (*cp)->dtor)
```

```
        self = (*cp)->dtor(self);
```

```
    free(self);
```

```
}
```


NEW / DELETE HA C

```
//main.c
#include "point.h"
#include "new.h"

int main(int argc, char **argv)
{
    void *p = new(Point, 1, 2);
    delete(p);

    return 0;
}
```

STATIC MEMBERS

```
// box.h
class Box {
public:
    Box(double l, double b, double h) :
        length(l), breadth(b), height(h)
    {
        ++objectCount;
    }

    ~Box() { --objectCount; }

    static int getObjectCount() { return objectCount; }

private:
    double length, height, breadth;

    static int objectCount;
};

//-----
// box.cpp
int Box::objectCount = 0;
```

Вызов функции:
Box::getObjectCount()

STATIC MEMBERS

```
// box.h
class Box {
public:
    Box(double l, double b, double h) :
        length(l), breadth(b), height(h)
    {
        ++objectCount;
    }

    ~Box() { --objectCount; }

    static int getObjectCount() { return objectCount; }

private:
    double length, height, breadth;

    static inline int objectCount = 0; // Начиная с C++17
};
```


INLINE

```
// foo.h  
class Foo {  
    // ...  
    int getX();
```

```
private:  
    int x;
```

```
};
```

```
inline int Foo::getX() {  
    return x;  
}
```

```
inline int max(int x, int y) {  
    return (x < y) ? y : x;  
}
```

```
Foo foo;
```

```
// Тот же код, что и в:  
//      int a = foo.x;  
int a = foo.getX();
```

Тело функции объявляется
в заголовочном файле!

```
// foo.h
class Foo {
    // ...
    int getX() { return x; }

private:
    int x;
};
```

Более короткая запись inline-методов

FUNCTION OVERLOADING

```
int sqrt(int n);
double sqrt(double d);
char *sqrt(char *);

void f() {
    int i      = sqrt(36);
    double d1  = sqrt(49.0);
    double d2  = sqrt(double(i*i));
    char *s    = sqrt("36");
}
```


```
//stack.h
class Stack{
    size_t size;
    /*...*/
public:
    Stack();
    ~Stack();

    int* top();
    const int* top() const;

    ...
};
```


CONSTRUCTOR OVERLOADING

```
//stack.h
class Stack{
    size_t size;
    /*...*/
public:
    Stack();
    Stack(const int* const arr, const size_t size);
    ~Stack();
    ...
};
```



Destructor is always one.

MANGLING («МАНГЛИНГ»)

Прототип	GNU C++	Microsoft Visual C++
<code>void h(int);</code>	<code>_Z1hi</code>	<code>?h@@YAXH@Z</code>
<code>void h(int, char);</code>	<code>_Z1hic</code>	<code>?h@@YAXHD@Z</code>
<code>void h(void);</code>	<code>_Z1hv</code>	<code>?h@@YAXXZ</code>

INHERITANCE

```
class Base {  
public:  
    Base();  
    void say();  
    virtual void sayVirtual();  
};
```

```
inline Base::Base()           { puts("Base"); }  
inline void Base::say()       { puts("Base::say"); }  
inline void Base::sayVirtual() { puts("Base::sayVirtual"); }
```

//-----

```
class Derived : public Base {  
public:  
    Derived()           { puts("Derived"); }  
    void say()          { puts("Derived::say"); }  
    void sayVirtual()   { puts("Derived::sayVirtual"); }  
};
```

```
Base *b = new Derived; // -> Base Derived  
b->say();               // -> Base::say  
b->sayVirtual();        // -> Derived::sayVirtual  
delete b;
```


ACCESS MODIFIERS

public	private	protected
Все, кто имеет доступ к описанию класса	Функции-члены класса	Функции-члены класса
	«Дружественные» функции	«Дружественные» функции
	«Дружественные классы»	«Дружественные классы»
		Функции-члены производных классов

```
class Base {  
    public:  
        Base();  
  
    protected:  
        int getSecret() { return secret; }  
  
    private:  
        int secret;  
};  
  
class Derived : public Base {  
    public:  
        void doSomething() {  
            int x = getSecret();  
            // ...  
        }  
  
        void error() {  
            secret++;    // ERROR  
        }  
};
```

STRUCT VS CLASS

```
struct X {  
    int a;           // public  
};
```

```
class Y {  
    int a;           // private  
};
```

Есть еще одно отличие в наследовании.



Попробуем реализовать наследование на С?

ОБЪЕКТНАЯ СИСТЕМА НА C

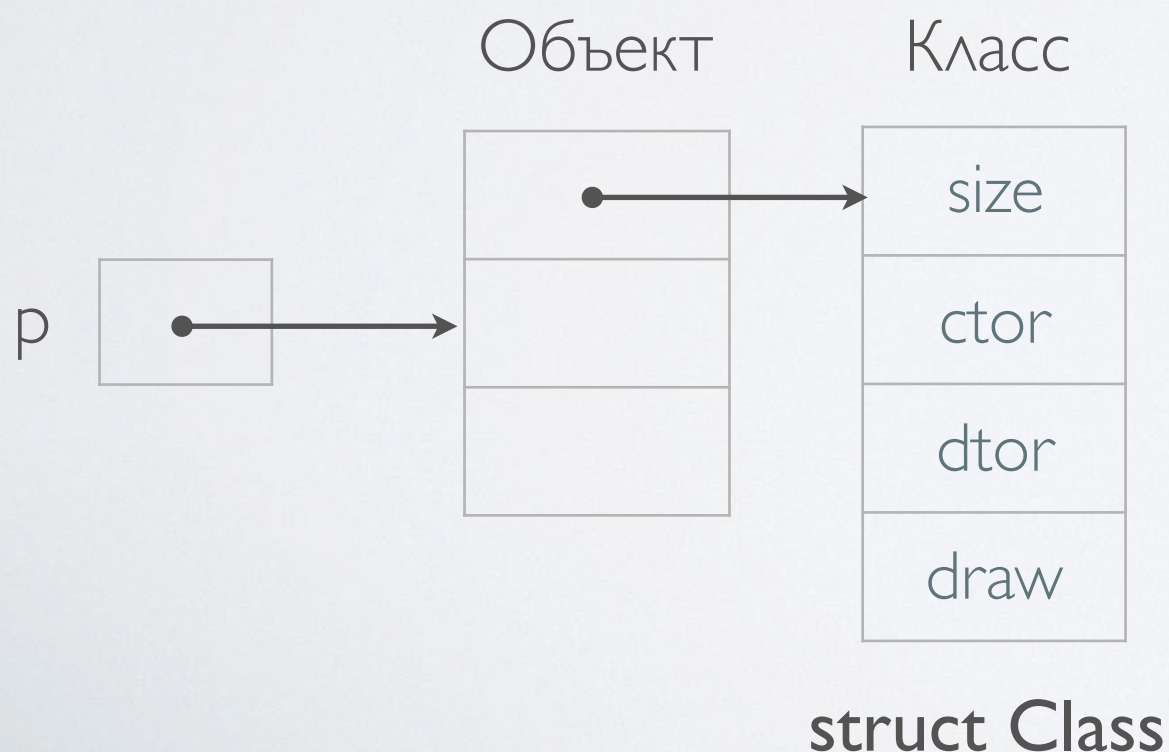
```
// new.h
struct Class {
    size_t size;
    void *(*ctor)(void *self, va_list *app);
    void *(*dtor)(void *self);
    void (*draw)(const void *self);
};

void *new(const void *class, ...);
void delete(void *item);
void draw(const void *self);
```

```
// new.c
void *new(const void *_class, ...)
{
    const struct Class *class = _class;
    void *p = calloc(1, class->size);
    assert(p);
    *(const struct Class **)p = class;

    if (class->ctor) {
        va_list ap;
        va_start(ap, _class);
        p = class->ctor(p, &ap);
        va_end(ap);
    }
    return p;
}

// продолжение следует ...
```



```
// new.c
```

```
void delete(void *self)
{
    const struct Class **cp = self;

    if (self && *cp && (*cp)->dtor)
        self = (*cp)->dtor(self);

    free(self);
}
```

```
void draw(const void *self)
{
    const struct Class *const *cp = self;
    assert(self && *cp && (*cp)->draw);
    (*cp)->draw(self);
}
```

```
// new.h
```

```
struct Class {
    size_t size;
    void *(*ctor)(void *self,
                  va_list *app);
    void *(*dtor)(void *self);
    void (*draw)(const void *self);
};

void *new(const void *class, ...);

void delete(void *item);

void draw(const void *self);
```



```
// point.c
```

```
void move(void *_self, int dx, int dy) {  
    struct Point *self = _self;  
    self->x += dx;  
    self->y += dy;  
}
```

Нединамический метод

```
static void *Point_ctor(void *_self, va_list *app) {  
    struct Point *self = _self;  
    self->x = va_arg(*app, int);  
    self->y = va_arg(*app, int);  
    return self;  
}
```

Конструктор Point

```
static void Point_draw(const void *_self) {  
    const struct Point *self = _self;  
  
    printf("\\".\"" at %d,%d\\n", self->x, self->y);  
}
```

```
static const struct Class _Point = {  
    sizeof(struct Point),    // size  
    Point_ctor,              // ctor  
    0,                       // dtor  
    Point_draw               // draw  
};
```

```
const void *Point = &_amp;_Point;
```

Деструктора нет

```
// point.h  
struct Point {  
    const void *class;  
    int x, y; /* координаты */  
};  
  
extern const void *Point;  
  
void move(void *_self,  
          int dx,  
          int dy);
```

```
// circle.h
struct Circle {
    const struct Point _;
    int rad;
};
```

«Круг — это такая жирная точка
с радиусом **rad**»

```
// circle.c
static void *Circle_ctor(void *_self, va_list *app) {
    struct Circle *self = ((const struct Class *)Point)->ctor(_self, app);
    self->rad = va_arg(*app, int);
    return self;
}

#define x(p) (((const struct Point *) (p)) -> x)
#define y(p) (((const struct Point *) (p)) -> y)

static void Circle_draw(const void * _self) {
    const struct Circle *self = _self;
    printf("circle at %d,%d rad %d\n", x(self), y(self), self->rad);
}

static const struct Class _Circle = {
    sizeof(struct Circle), Circle_ctor, 0, Circle_draw
};

const void *Circle = &_Circle;
```



```

#include "point.h"
#include "circle.h"
#include "new.h"

int main(int argc, char **argv)
{
    void *p;
    while (*++argv) {
        switch (**argv) {
            case 'p':
                p = new(Point, 1, 2);
                break;
            case 'c':
                p = new(Circle, 1, 2, 3);
                break;
            default:
                continue;
        }

        draw(p);
        move(p, 10, 20);
        draw(p);
        delete(p);
    }

    return 0;
}

```

```

$ circles p c
"." at 1,2
"." at 11,22
circle at 1,2 rad 3
circle at 11,22 rad 3

```


КОНЕЦ ВТОРОЙ ЛЕКЦИИ

```
Lection::~~Lection() { }
```