

Программирование на языке C++

Лекция 3

Конструкторы и деструкторы

Александр Смаль

Конструкторы

Конструкторы — это методы для инициализации структур.

```
struct Point {  
    Point() {  
        x = y = 0;  
    }  
    Point(double x, double y) {  
        this->x = x;  
        this->y = y;  
    }  
    double x;  
    double y;  
};
```

```
Point p1;  
Point p2(3,7);
```

Список инициализации

Список инициализации позволяет проинициализировать поля до входа в конструктор.

```
struct Point {  
    Point() : x(0), y(0)  
    {}  
    Point(double x, double y) : x(x), y(y)  
    {}  
  
    double x;  
    double y;  
};
```

Инициализации полей в списке инициализации происходит в *порядке объявления полей* в структуре.

Значения по умолчанию

- Функции могут иметь значения параметров *по умолчанию*.
- Значения параметров по умолчанию нужно указывать в *объявлении функции*.

```
struct Point {  
    Point(double x = 0, double y = 0)  
        : x(x), y(y)  
    {}  
    double x;  
    double y;  
};
```

```
Point p1;  
Point p2(2);  
Point p3(3,4);
```

Конструкторы от одного параметра

Конструкторы от одного параметра задают *неявное* пользовательское преобразование:

```
struct Segment {  
    Segment() {}  
    Segment(double length)  
        : p2(length, 0)  
    {}  
    Point p1;  
    Point p2;  
};
```

```
Segment s1;  
Segment s2(10);  
Segment s3 = 20;
```

Конструкторы от одного параметра

Для того, чтобы запретить *неявное* пользовательское преобразование, используется ключевое слово `explicit`.

```
struct Segment {  
    Segment() {}  
    explicit Segment(double length)  
        : p2(length, 0)  
    {}  
    Point p1;  
    Point p2;  
};
```

```
Segment s1;  
Segment s2(10);  
Segment s3 = 20; // error
```

Конструкторы от одного параметра

Неявное пользовательское преобразование, задаётся также конструкторами, которые могут принимать один параметр.

```
struct Point {  
    explicit Point(double x = 0, double y = 0)  
        : x(x), y(y)  
    {}  
    double x;  
    double y;  
};
```

```
Point p1;  
Point p2(2);  
Point p3(3,4);  
Point p4 = 5; // error
```

Конструктор по умолчанию

Если у структуры нет конструкторов, то конструктор без параметров, *конструктор по умолчанию*, генерируется компилятором.

```
struct Segment {  
    Segment(Point p1, Point p2)  
        : p1(p1), p2(p2)  
    {}  
    Point p1;  
    Point p2;  
};
```

```
Segment s1; // error  
Segment s2(Point(), Point(2,1));
```


Особенности синтаксиса C++

“Если что-то похоже на объявление функции, то это и есть объявление функции.”

```
struct Point {  
    explicit Point(double x = 0, double y = 0)  
        : x(x), y(y) {}  
    double x;  
    double y;  
};
```

```
Point p1;    // определение переменной  
Point p2();  // объявление функции  
  
double k = 5.1;  
Point p3(int(k)); // объявление функции  
Point p4((int)k); // определение переменной
```

Деструктор

Деструктор — это метод, который вызывается при удалении структуры, генерируется компилятором.

```
struct IntArray {  
    explicit IntArray(size_t size)  
        : size(size)  
        , data(new int[size])  
    { }  
  
    ~IntArray() {  
        delete [] data;  
    }  
  
    size_t size;  
    int * data;  
};
```

Время жизни

Время жизни — это временной интервал между вызовами конструктора и деструктора.

```
void foo()
{
    IntArray a1(10); // создание a1
    IntArray a2(20); // создание a2
    for (size_t i = 0; i != a1.size; ++i) {
        IntArray a3(30); // создание a3
        ...
    } // удаление a3
} // удаление a2, потом a1
```

Деструкторы переменных на стеке вызываются в обратном порядке (по отношению к порядку вызова конструкторов).