# Лабораторная работа №2 «Конструкторы и деструкторы»

## Цель

Научиться описывать конструкторы и деструкторы для класса, понимать различия между конструкторами. Понимать, в каких случаях могут быть вызваны конструкторы копирования, понимать время жизни объекта.

## Теоретическая информация

### Конструктор. Виды конструкторов

Конструктор – выделяет память для объекта и инициализирует данные-члены класса.

- имя конструктора совпадает с именем класса;
- конструктор не имеет возвращаемого значения (даже void);
- классу без конструктора предоставляется конструктор по умолчанию;
- при явно описанном конструкторе конструктор по умолчанию не генерируется;
- конструкторы могут быть перегружены;
- не могут быть описаны с ключевыми словами virtual, static, const, mutable, volatile;
- не могут явно вызываться в программе.

Конструктор представляет собой обычную функцию, имя которой совпадает с именем класса, в котором он объявлен и используется. Количество и имена фактических параметров в описании функции конструктора чаще зависят от числа полей, которые будут инициализированы при объявлении объекта (экземпляра) данного класса. Кроме отмеченной формы записи конструктора в программах на C++ можно встретить и форму записи конструктора в следующем виде:

```
kls(int A, int B, int C) :  a(A), b(B), c(C) { }
```

В этом случае после двоеточия перечисляются инициализируемые данные и в скобках - инициализирующие их значения (точнее, через запятую перечисляются конструкторы объектов соответствующих типов). Возможна комбинация отмеченных форм.

Наряду с перечисленными выше формами записи конструктора существует конструктор, либо не имеющий параметров, либо все аргументы которого заданы по умолчанию – конструктор по умолчанию:

kls(){ } это, для примера выше, аналогично kls() : a(), b(), c() { }

kls(int=0, int=0, int=0){ } это аналогично kls() : a(0), b(0), c(0) { }

Каждый класс может иметь только один конструктор по умолчанию. Конструктор по умолчанию используется при создании объекта без инициализации его, а также незаменим при создании массива объектов. Если при этом конструкторы с параметрами в классе есть, а конструктора по умолчанию нет, то компилятор зафиксирует синтаксическую ошибку.

В классе может быть объявлено (и определено) несколько конструкторов. Их объявления должны различаться списками параметров. Такие конструкторы по аналогии с функциями называются перегруженными (или совместно используемыми). Транслятор различает перегруженные конструкторы по спискам параметров. В этом смысле конструктор не отличается от обычной функции-члена класса:

ComplexType(double rePar, double imPar); /* Объявление… */

ComplexType(double rePar, double imPar) { /*…*/ } /*Определение…*/

Конструктор вызывается не для объекта класса, как другие функции-члены, а для области памяти. Для её преобразования ("превращения") в объект класса.

**Деструктор**

Противоположным по отношению к конструктору является деструктор - функция, приводящая к разрушению объекта соответствующего класса и возвращающая системе область памяти, выделенную конструктором.

Деструктор имеет имя, аналогичное имени конструктора, но перед ним ставится знак ~:

~kls(void){}  или  ~kls(){}  // функция-деструктор

Деструктор – вызывается автоматически при разрушении объекта, его задача - освободить память и корректно уничтожить объект.

- имя деструктора также совпадает с именем класса но предваряется символом тильда "~" (~имя_класса);
- деструктор не имеет ни какого возвращаемого значения;
- не может быть описан как static, const, mutable, volatile;
- без явного задания генерируется деструктор по умолчанию;
- могут быть описаны как virtual - у всех производных классов деструкторы автоматически будут виртуальными;
- могут явно вызываться.

**Конструктор копирования**

Частным случаем конструктора является конструктор копирования, у которого в аргументах передается ссылка на другой объект того же класса.

Необходимость использования конструктора копирования вызвана тем, что объекты наряду со статическими могут содержать и динамические данные. В тоже время, например, при передаче объекта в качестве параметра функции в ней создается локальная (в пределах функции), копия этого объекта. При этом указатели обоих объектов будут содержать один и тот же адрес области памяти. При выводе локального объекта из поля видимости функции для его разрушения вызывается деструктор. В функцию деструктора входит также освобождение динамической памяти, адрес которой содержит указатель. При окончании работы программы (при вызове деструкторов), производится повторная попытка освободить уже освобожденную ранее память. Это приводит к ошибке. Для устранения этого в класс необходимо добавить конструктор копирования, который в качестве единственного параметра получает ссылку на объект класса. Общий вид конструктора копирования имеет следующий вид:

имя_класса (const имя_класса  & );

В этом конструкторе выполняется выделение памяти и копирование в нее информации из объекта получаемого по ссылке. Таким образом, указатели для обоих объектов содержат разные адреса памяти, и при вызове деструктора выполняется освобождение соответствующей каждому из объектов памяти.

```
#include <iostream.h>
#include <stdlib.h>
#define n 3
//------------------     объявление     konstr_copy.h
class cls
{  char *str;
   int dl;
        // другие данные класса
  public:
   cls ();           //  конструктор по умолчанию
   cls(cls &);    //  копирующий конструктор
   ~cls();         //  деструктор
        // другие методы класса
};
//------------------    реализация     konstr_copy.cpp
#include "konstr_copy.h"
cls::cls ()
{ dl=10;
  str=new char[dl];
}
cls::cls(cls & obj1)          // копирующий конструктор из obj1 в  obj
{ dl=obj1.dl;                 // копирование длины строки
  str=new char[dl];           // выделение памяти “под” отроку длинной dl
  strcpy(str,obj1.str);       //  копирование строки
}
```

```
cls::~cls()
{ delete [] str;
   cout<<"деструктор"<<endl;
}

void fun(cls obj1)
{  // код функции
    cout<<" выполняется функция "<<endl;
}

void main(void)
{ cls obj;
  //   . . .
   fun(obj);
  // . . .
}
```

Если для класса конструктор копирования явно не описан, то компилятор сгенерирует его. При этом значения компоненты-данного одного объекта будут скопированы в компоненту-данное другого объекта. Это допустимо для объектов простых классов и недопустимо для объектов, имеющих динамические компоненты-данные (конструируются с использованием операторов динамического выделения памяти). Таким образом, даже если в классе не используются динамические данные, желательно явно описывать конструктор копирования.

**Задание**

1. Дополнить и при необходимости модифицировать приложение, разработанное согласно варианта лабораторной работы №1.
2. Определить в классах следующие конструкторы: без параметров, с параметрами, копирования, перемещения.

3. Определить в классе деструктор.
4. В каждом конструкторе и деструкторе выдавать сообщение, показывающее, какой именно конструктор или деструктор был вызван.
5. Дополнить основной класс методом с параметрами, внутри которого должен создаваться объект дополнительного класса с помощью конструктора с параметрами и добавляться в массив объектов дополнительного класса.
6. Дополнить основной класс методом с двумя параметрами (ссылка на объект дополнительного класса и количество), внутри которого должны создаваться объекты дополнительного класса с помощью конструктора копирования (создается необходимое количество копий указанного объекта) и заноситься в массив объектов дополнительного класса.
7. Дополнить/модифицировать основное тело программы таким образом, чтобы продемонстрировать использование всех конструкторов.
8. Сделать выводы.

**Варианты**

Варианты соответствуют лабораторной работе №1.

**Итоговый отчет**

Укажите в отчете:

1. Тему лабораторной работы, свою фамилию и имя, группу, вариант задания;
2. Листинг программы;
3. Скриншоты результатов выполнения программы;
4. Краткие пояснения к алгоритму работы;
5. Выводы.

**Дополнительные вопросы**

1. Какие конструкторы класса существуют?
2. Как создавать конструктор с параметрами?

3. В чем особенность передачи объекта методу в качестве параметра?
4. Когда будет разрушен объект, созданный внутри метода?
5. Когда разрушаются объекты, созданные динамически?