Т.2.7.3. Конструкторы и деструкторы.

Конструктор

Конструктор класса – это метод класса, вызываемый при создании объекта и служащий для задания данным объекта начальных значений.

Конструктор **имеет то же имя, что и класс и не имеет типа.**

Конструкторы не наследуются, однако, стандартные конструкторы и конструкторы копирования при необходимости создаются транслятором.

Стандартным конструктором для класса X является такой конструктор класса X, который можно вызывать без параметров. Стандартный конструктор для класса X будет создан только тогда, когда для класса X не описано ни одного конструктора.

В отличии от конструктора, **деструктор (от слова destruct — разрушать)** — специальный метод класса, который служит для уничтожения элементов класса. Чаще всего его используют тогда, когда в конструкторе, при создании объекта класса, динамически был выделен участок памяти и необходимо эту память очистить, если эти значения уже не нужны для дальнейшей работы программы.

Конструктор

Указывать конструктор не обязательно; если конструктор не объявлен, при создании объекта будет вызван **стандартный конструктор**, который просто создаст данные-члены объекта в памяти, при этом в них будет находиться «мусор», поэтому желательно всегда явно указывать конструктор, задающий начальные значения полям объекта.

В случае если аргумент конструктора – другой объект, говорят о **конструкторе копирования**.

Аргумент обязательно должен передаваться по ссылке, иначе при вызове конструктора копирования для передачи аргумента будет создаваться копия объекта при помощи того же конструктора копирования, таким образом, произойдёт рекурсивный вызов функции без возможности выхода из неё.

Конструктор

В конструкторе для краткости может применяться **список инициализации**, который указывается через двоеточие сразу после списка параметров конструктора. В нём происходит инициализация параметров следующим образом:

```
class MyClass
{
public:
    MyClass() : имяПеременной(значениеПоУмолчанию), ... {тело конструктора }
};
```

Конструктор

```
class date {
 int month, day, year;
    public:
 date(int, int, int); // день месяц год
 date(char*); // дата в строковом представлении
 date(); // дата по умолчанию: сегодня
    };

date mart("march 22, 2022");
date guy(22, 3, 2022);
date now; // инициализируется по умолчанию
```

Конструктор

```
class MyClass //Объявили класс MyClass
   {
    int a;
    public:
      void Show()
{  cout<<a; }

   MyClass()
{   cout<<"Wwedi a: ";
   cin>>a;
  }
   };
```

Виды конструкторов

Первый – пустой конструктор, вызывающийся всегда, когда явно не указан вызов другого конструктора. В нём при помощи списка инициализации цвет яблока задаётся как белый, а вес – 100 грамм.

- Далее идёт конструктор, в который передаются все параметры нового яблока (вес и цвет).
- Последний конструктор – конструктор копирования, он копирует данные из другого объекта.

```
class Apple
{
public:
    unsigned int color;
    unsigned int weight;
    Apple() : color(0xffffffff), weight(100) {}
    Apple(unsigned int _color, unsigned int _weight) : color(_color), weight(_weight) {}
    Apple(Apple &otherApple) : color(otherApple.color), weight(otherApple.weight) {}
};
```

Важно:

- Конструктор и деструктор, мы всегда объявляем в разделе public;
- при объявлении конструктора, тип данных возвращаемого значения не указывается, в том числе — void;
- у деструктора также нет типа данных для возвращаемого значения, к тому же деструктору нельзя передавать никаких параметров;
- имя класса и конструктора должно быть идентично;
- имя деструктора идентично имени конструктора, но с приставкой ~ ;
- В классе допустимо создавать несколько конструкторов, если это необходимо. Имена будут одинаковыми. Компилятор будет их различать по передаваемым параметрам (как при перегрузке функций). Если мы не передаем в конструктор параметры, он считается конструктором по умолчанию;
- В классе может быть объявлен только один деструктор;

Пример

```
class Complex
{ int real;
 int image;
 public:
void show();
 Complex();
 Complex(int re);
};
Complex::Complex():real(0),image(0)
{};
Complex::Complex(int re):real(re),image(0)
{};
```

Пример

```
int main()
{
   Complex a;
   Complex  b(6);
   a.show();
   b.show();
  return 0;
}
```