ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Описание класса

```
class <имя>
{
   [ private: ]
   <описание скрытых элементов>
   public:
   <описанве доступных элементов>
   };       // Описание заканчивается точкой с запятой
```

Поля класса

- могут иметь любой тип, кроме типа этого же класса
- могут быть описаны с модификатором const
- могут быть описаны с модификатором static
- не могут быть описаны с модификаторами auto, extern. Register
- инициализация полей при описании не допускается

Особенности локального класса

- внутри класса можно использовать типы, static и extern переменные, внешние функции, элементы перечислений из области, в которой он описан
- запрещается использовать автоматические переменные из этой области
- локальный класс не может иметь статических элементов
- методы этого класса могут быть описаны только внутри класса
- если один класс вложен в другой, то они не имеют каких-либо особых прав доступа к элементам друг друга и могут обращаться к ним только по общим правилам

Пример

```
class monstr
{
        int health, ammo;
      public:
        monstr{int he = 100, int am = 10) {health = he; ammo = am;}
      void draw(int x, int y, int scale, int position);
      int get_health()(return health;}
      int get_ammo(){return amo;}
};

void monstr::draw(int x, int y, int scale, int position)
{
/* тело метода */
}

inline int monstr::get_ammo()
{
      return ammo;
}
```

Описание объектов

```
monstr Vasia; // Обьект класса monstr с параметрами по умолчанию
monstr Super(200, 300);  // Обьект с явной инициализацией
monstr stado[100]; //Массив обьектов с параметрами по
    умолчанию
monstr *beavis = new monstr (10); // Динамический объект
                                  //(второй параметр задается по умолчанию)
monstr &butthead = Vasia;    // Ссылка на объект
```

Доступ к элементам объекта

. (точка)

->

Например:

```
Int n = Vasia.get_ammo();
stado[5].draw;
Cout << beavis -> get_health();
```

Указатель this

```
monstr & the_best(monstr &M)
{
    if( health > M.health) return *this;
    return M;
}
…
monstr Vasia(50), Super (200);
// Новый объект Best инициализируется
// значениями полей Super

monstr Best = Vasia.the_best(Super);
```

```
Void cure(int health. Int ammo)
{
    this -> health += health; // Использование this
    monstr :: ammo += ammo // Использование операции ::
```

Конструкторы

- не возвращает значение
- нельзя получить указатель на конструктор
- класс может иметь несколько конструкторов
- конструктор без параметров – конструктор по умолчанию
- компилятор создает автоматически

Конструкторы

- конструкторы не наследуются
- нельзя использовать модификаторы const, virtual, static
- конструкторы глобальных объектов вызываются до вызова функуии main

Конструкторы

- Локальные объекты создаются как только становится активной область их действия
- конструктор вызывается, если в программе встретилась какая-либо из синтаксических конструкций:

```
Имя_класса имя_объекта [(список параметров)];
// Список параметров не должен быть пустым
имя_класса (список параметров);
//Создается объект без имени (список может быть пустым)
имя_класса имя_объекта  = выражение;
// Создается объект без имени и копируется
```

Конструкторы

Примеры:

monstr Super(200,300), Vasia(50), Z;

monstr X=monstr(1000);

monstr Y=500;

Конструктор копирования

T::T(const T&) { … /* Тело конструктора*/}

где T – имя класса

- при описании нового объекта с инициализацией другим объектом;
- при передаче объекта в функцию по значению;
- при возврате объекта из функции.

```
monstr::monstr(const monstr &M)
{
    if (M.name)
    {
      name=new char [strlen(M.name)+1];
      strcpy(name, M.name);
    }
    else name=0;
    health =M.health; ammo=M.ammo; skin=M.skin;
}
…
monstr  Vasia(blue);
monstr  Super=Vasia; //Работает конструктор копирования
monstr  *m=new monstr (“Orc”);
monstr  Green =*m; //Работает конструктор копирования
```

Статические элементы класса

- статические поля
- статические методы

Дружественные функции и классы

- дружественная функция
  - Объявляется внутри класса со словом friend
  - Может бытьобычной или методом другого ранее определенного класса
  - Может дружить с несколькими классами одновременно
- дружественный класс

Деструкторы

- для локальных объектов
- для глобальных
- для объектов заданных через указатель (delete)

Деструкторы

Имя задается с помощью тильды ~

Деструктор:

- не имеет аргументов и возвращаемого значения
- не может быть объявлен как const или static
- не наследуется
- может быть виртуальным