Классы

Описание класса

Класс – абстрактный тип данных, определяемых программистом, модель реального объекта в виде данных и функций для работы с ними.

Данные класса называются *полями* (по аналогии с полями структуры), а функции класса — *методами* . Поля и методы называются *элементами класса*. Описание класса в первом приближении выглядит так:

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

Пример описания класса игрового персонажа

```
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 ammo;}
};

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

Пример определения метода вне класса

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

Описание объектов (экземпляров класса)

```
monstr Vasia;                 // Объект класса monstr с параметрами по умолчанию
monstr Super(200, 300);       // Объект с явной инициализацией
monstr stado[100];            // Массив объектов 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 объектов

При обращении через имя объекта используется оператор точка (.), при обращении через указатель – оператор стрелка(–>):

```
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);
```

Конструкторы

Конструктор предназначен для инициализации объекта и вызывается автоматически. Пример класса с несколькими конструкторами:

```
enum color {red, green, blue};  // Возможные значения цвета
class monstr{
      int health, ammo;
      color skin;
      char *name;
   public:
      monstr(int he = 100, int am  = 10);
      monstr(color sk);
      monstr(char * nam);
      int get_health(){return health;}
      int get_ammo(){return ammo;}
   …
};
//-------------------------------
monstr::monstr(int he, int am){
   health = he; ammo = am; skin = red; name = 0;
}
//-------------------------------
monstr::monstr(color sk){
   switch (sk){
      case red  : health = 100; ammo = 10; skin = red;    name = 0; break;
      case green: health = 100; ammo = 20; skin = green;  name = 0; break;
      case blue : health = 100; ammo = 40; skin = blue;   name = 0; break;
   }
}
```

Конструктор копирования

Конструктор копирования — это специальный вид конструктора, получающий в качестве единственного параметра ссылку на объект этого же класса:

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

где T — имя класса.

Пример определения конструктора копирования

```
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 ("Ork");
monstr Green = *m;             // Работает конструктор копирования
```

Статические поля

Применяются для хранения данных общих для всех объектов класса

```
class A{
   public:
      static int count;  // Объявление в классе
};
…
int A::count;             // Определение в глобальной области
                          // По умолчанию инициализируется нулем
// int A::count = 10;     Пример инициализации произвольным значением
```

Статические поля доступны как через имя класса, так и через имя объекта:

```
A *a, b;
…
cout << A::count << a->count << b.count;
// Будет выведено одно и то же
```

Статические методы

```
class A{
      static int count;  // Поле count – скрытое
   public:
      static void inc_count(){ count++; }
      …
};
…
A::int count;          // Определение в глобальной области
void f(){
   A a;
   // a.count++ – нельзя, поле count скрытое
   // Изменение поля с помощью статического метода:
   a.inc_count();      // или A::inc_count();
}
```

Дружественные функции и классы

Дружественная функция

Применяется для доступа извне к скрытым полям класса

```
class monstr;   // Предварительное объявление класса
class hero{
   public:
      void kill(monstr &);
   …
};
class monstr{
   …
   friend int steal_ammo(monstr &);
   friend void hero::kill(monstr &);
   // Класс hero должен быть определен ранее
};
int steal_ammo(monstr &M){return --M.ammo;}
void hero::kill(monstr &M){M.health = 0; M.ammo = 0;}
```

Дружественный класс

```
class hero{
   …
   friend class mistress;
}
class mistress{
   …
   void f1();
   void f2();
}
```

Деструкторы

Деструктор – метод освобождения памяти, занимаемой объектом.

```
monstr::~monstr() {delete [] name;}
monstr *m; …
m -> ~monstr();
```