Классы. Описание класса

Константные методы.
Статические компоненты класса

Перегрузка функций

```
int fun(int i, int n=10)
{
return i*n;
}
double fun(double d, int n=10)
{
return d*double(n);
}
```

```
int i=20;
double d=4.44;
cout << fun(i) << endl;
cout << fun(i,20) << endl;
cout << fun(d) << endl;
cout << fun(d,5.55) << endl;
```

Классы. Описание класса

Константные методы класса

Рассмотрим пример.

```
 class Test
{
 int test_int;
 double test_double;
public:
 Test(){};
Test(int i, double d): test_int(i), test_double(d){};
```

Классы. Описание класса

```
void Out_const() const
    {
        cout << " Int: " << test_int << ' ' << " Double: " <<
        test_double << endl;
    }
    void Out()
    {
     cout << " Int: " << test_int << ' ' << " Double: " <<
    test_double << endl;
    }
    };
```

Здесь мы видим два одинаковых метода void Out_const() const и void Out (), первый из которых объявлен как константный. Его можно использовать по отношению к любым объектам класса, в том числе и к константным. Второй метод может использоваться только к не константным объектам.

Классы. Описание класса

Например,

```
Test tst(10, 3.45);
const Test tst_const(20, 7.89);
tst. Out() ;        // ok
tst. Out_const();  // ok
tst_const.Out();   // error
tst_const. Out_const();     // ok
```

Классы. Описание класса

sСуть ошибки в следующем:

test_const_meth.cpp(28): error C2662: Test::Show: невозможно преобразовать указатель "this" из "const Test" в "Test

Еще одно ограничение на константный метод – он не может менять значение полей класса.

Классы. Описание класса

Например,

```
void Out_const() const
{
test_int = 333L;   // error
cout << " Int: " << test_int << ' ' << " Double: "
<< test_double << endl;
}
```

Классы. Описание класса

Приведет к ошибке вида:

\test_const_meth.cpp(13): error C3490: "test_int" не может быть изменен, поскольку доступ к нему осуществляется через константный объект

Классы. Описание класса

Значение полей константного объекта менять нельзя. В отдельных случаях, отдельные поля все-таки изменить можно, описав их со спецификатором mutable.

Пересмотрим класс Test:

```
class Test
{
 mutable int test_int;    // изменяемое поле
 double test_double;
public:
Test(){};
```

Классы. Описание класса

```
Test(int i, double d): test_int(i), test_double(d){};
    void Out_const() const
     {
    cout << " Int: " << test_int << ' ' << " Double: " <<
    test_double << endl;
    }
    void Setter(int i) const   // метод изменяющий поле
    {
     test_int = i;
    }
    };
```

Классы. Описание класса

Пример использования:

```
const Test tst_const(20, 7.89);
tst_const.Out_const();
tst_const.Setter(200);
tst_const.Out_const();
```

Поле test_int объекта tst_const поменяет свое значение, попытка изменить значение другого объекта завершится ошибкой.

Безопасные функции относятся к привилегированным составляющим класса.

Классы. Описание класса

Статические компоненты класса

Поле класса, объявленное со спецификатором static, называется статическим полем. Такое поле является общим для всех объектов данного класса и существует даже тогда, когда не создано ни одного объекта данного класса.

Инициализация статического поля глобального класса осуществляется с помощью отдельного инициализирующего объявления.

Классы. Описание класса

Статические поля могут выполнять роль флажков условия, указателями на программу обработки ошибок для класса, или полей общих для всех объектов данного типа. Статическое поле хранится не в объекте (экземпляре) класса, а в сегменте данных проекта.

Важное замечание (напоминание): локальный класс не может иметь своих собственных статических компонентов, но может использовать таковые из ближайшего окружения.

Классы. Описание класса

Статические поля глобальных классов по умолчанию инициализируются нулем соответствующего типа.

Составляющие функции класса также могут быть описаны со спецификатором static. В отличие от остальных составляющих функций, статические функции могут обращаться только к статическим полям класса, а также к переменным и функциям, объявленных вне класса.

Классы. Описание класса

Для дальнейших рассуждений рассмотрим еще один класс:

```
class Students
{
    string Name;
    int Age;
public:
    Students(){};
    Students(string name, int age):Name(name), Age(age){};
    void Show()
    {
    cout << " Name: " << Name << ' ' << " Age: " << Age << endl;
    }
};
```

Классы. Описание класса

Объявим несколько объектов данного класса:

Students st_1("Ivan",20), st_2("Mary",20), st_3("Peter", 22);

Размер памяти, необходимой для хранения объекта типа Students – 36 байт. Под каждый из объявленных объектов будет выделено ровно 36 байт. Попробуем представить себе «разрез» объектов внутри памяти машины.

Классы. Описание класса

st_1	
Иван	32 бита
20	4 бита

st_2	
Mary	32 бита
20	4 бита

st_3	
Peter	32бита
22	4 бита

Классы. Описание класса

Поля Name и Age будут выделены абсолютно всем объектам данного класса, сколько бы объектов не было объявлено. Несложно заметить, что значения этих полей индивидуально для каждого объекта.

Классы. Описание класса

Такие поля в дальнейшем мы будем именовать *экземплярными* полями или полями одного экземпляра. Они индивидуальны для каждого объекта и хранятся внутри объектов на все время существования объекта.

Что может общим для всех нескольких студентов?

Классы. Описание класса

Объединяющим звеном для студентов может быть название (номер) группы, название факультета, специальности и многие другие моменты. Видимо их имеет смысл описать с помощью статических полей.

Классы. Описание класса

Пересмотрим еще раз класс Students

```
class Students
{
    string Name;
    int Age;
public:
    Students(){};
    Students(string name, int age):Name(name),
    Age(age){};
```

Классы. Описание класса

```
    static int Group;  // статическое поле
    void Show()
    {
    cout << " Name: " << Name << ' ' << " Age: " <<
    Age << endl;
    }
};
```

Классы. Описание класса

Далее, вне тела какого-либо блока (в глобальной области) делаем инициализацию статического поля:

```
int Students::Group = 4120;
```

Попытка сделать объявление внутри блока, например, в теле функции, приведет к ошибке. Еще одно важное замечание – со статическим полем можно работать даже в отсутствии объекта данного типа.

Классы. Описание класса

Пример работы со статическим полем:

```
Students st_1("Ivan",20);
// без участия объекта
cout << Students::Group << endl;
// через объект класса
cout << st_1.Group << endl;
```

Подобное обращение возможно, поскольку поле Group расположено в области public.

Классы. Описание класса

Следующий момент связан с тем, посредством каких методов можно работать со статическими полями.

Значения статических полей можно менять с помощью абсолютно любых составляющих методов класса и, кроме того, они доступны дружественным функциям класса.

Однако, для работы со статическими полями предназначены статические составляющие класса.

Классы. Описание класса

Они отличаются от обычных методов класса ключевым словом static.

Важное замечание: статические методы не могут работать с нестатическими полями класса (!).

Классы. Описание класса

Введем статический метод:

```
static void Show_static()
{
    cout << " Group: " << Group << endl;
    //cout << Name << ' ' << Age << endl;
}
```

Попытка: `cout << Name << ' ' << Age << endl;`
приведет к ошибке, в частности
(22): error C2597: недопустимая ссылка на нестатический член "Students::Name"

Классы. Описание класса

К статическому методу класса также можно обратиться даже без объявленных объектов данного класса:

```
Students::Show_static();
```

Классы. Описание класса

Не все составляющие функции класса могут быть объявлены как статические, например, конструкторы, деструкторы, некоторые перегружаемые операторы.

А отдельные операции, в частности, операции new и delete по умолчанию считаются статическими.

Статические методы не могут быть константными и виртуальными.

Классы. Описание класса

Особенности статических полей класса:

- память под статическое поле класса выделяется один раз при его инициализации не зависимо от числа объявленных объектов;
- статические поля доступны как через имя класса, так и через имя объекта;
- на статическое поле распространяется действие спецификатора private;

Классы. Описание класса

- память, занимаемая статическим полем, не учитывается при определении размера объекта с помощью операции sizeof.

Особенности статических методов (функций) класса:

- не могут воздействовать на нестатические поя касса;
- статические методы не могут быть константными и виртуальными.

Классы. Описание класса

***Конструкторы класса**

Инициализация полей объекта производится с помощью составной функции класса, называемой конструктором. Имя этой функции, задаваемой в неявно привилегированном виде, идентично имени класса, а ее идентификатор, как правило, перегружен. Это дает возможность для объектов данного класса

Классы. Описание класса

применять инициализаторы с разными типами аргументов и в разных количествах.

Конструктором может быть практически любая функция класса при следующих ограничениях: конструктор класса не содержит формального параметра типа класса, в определении конструктора не содержится определение типа результата (даже типа void), а если в теле содержится оператор return, он не использует выражений.

Классы. Описание класса

Если для какого-либо класса конструктор не определен, то для него неявно (автоматически) определяется конструктор без параметров (по умолчанию).

Основные свойства конструкторов:

- конструктор не возвращает значение, даже на тип void. Нельзя получить указатель на конструктор (!);
- конструктор, вызываемый без параметров, называется конструктором по умолчанию;

Классы. Описание класса

- конструкторы не наследуются;
- конструкторы нельзя описывать с модификаторами const, virtual и static;
- параметры конструктора могут иметь любой тип, кроме типа определяемого класса. Можно задавать параметры по умолчанию (только один конструктор);

Классы. Описание класса

- если в классе не задано ни одного конструктора, автоматически создается конструктор по умолчанию. В случае присутствия константных, ссылочных и объектных полей необходим конструктор со списком инициализации;

Классы. Описание класса

- конструкторы глобальных объектов вызываются до вызова функции main. Локальные объекты создаются, как только становится активной область их действия. Конструктор вызывается и при создании временного объекта, например, при передаче объекта из функции.

Классы. Описание класса

- конструктор вызывается, если в программе встретится какая-либо из синтаксических конструкций:

имя_класса имя_объекта [(сп. параметров)];

имя_класса (сп. параметров);

имя_класса имя_объекта = выражение;

Классы. Описание класса

Рассмотрим пример класса с несколькими конструкторами.

```
class Test
{
    int test_int;
    string test_string;
public:
    Test()
{ cout << " Конструктор без параметров " <<
    endl;}
```

Классы. Описание класса

```
Test(int i, string s)
{
test_int = i;
test_string = s;
cout << " Конструктор с параметрами " << endl;
}
Test(int i)
{
test_int = i;
cout << " конструктор преобразования " << endl;
}
```

Классы. Описание класса

```
    void Show() const
    {
    cout << " Int: " << test_int << " String: " <<
    test_string << endl;
    }
};
```

Классы. Описание класса

Возможные способы объявления и инициализации объектов класса Test:

```
Test tst_1;
Test tst_2(10);
Test tst_3(20,"String");
Test(30);
```

В последнем случае создается безымянный объект со значением поля test_int = 30.

Классы. Описание класса

Рекомендуется испытать программу и оценить результаты, это важно, потому что в разных случаях будут работать совершенно разные конструкторы.

Классы. Описание класса

Конструктор со списком инициализации

Среди конструкторов следует выделить особенный случай конструктора со списком инициализации.

Классы. Описание класса

Классы. Описание класса

Классы. Описание класса

Классы. Описание класса

Классы. Описание класса

Классы. Описание класса

Классы. Описание класса