# лекция 14

**Тема лекции:** Динамическая информация о типе. Указатели на функции-члены класса. Статические члены класса. Размещение объектов по заданным адресами. Инициализация членов объекта класса. Вложенные объявления в классе.

***Динамическая информация о типе***

Динамическая информация о типе предназначена для предоставления информации о тип объекта во время выполнения программы. Можно использовать ее для сравнения типов объектов и для доступа к строкам, в которых они описываются. Оператор *typeid* возвращает объект класса *Type\_info,* описывающий тип объекта. Можно использовать *typeid* как для простых объектов встроенных типов, так и для классов, структур, массивов, собственных типов данных для сравнения на эквивалентность (равен / не равно) или в алфавитном порядке. для использования *typeid* нужно включить в программу заголовочный файл *TYPEINFO.H.* В классе *Typeinfo* есть функции-члены, к которым можно обратиться. К примеру, *name ()* - функция, предназначенная для доступа к имени объекта:

int count;

cout << typeid (count) .name ();

Оператор вывода отобразит *int* - имя типа данных переменной *count.*

Сравнение типов данных объектов:

int i;

int j;

if (typeid (i) == typeid (j))

cout << "одинаковые типы";

Функция *before ()* вызывается для сравнения типов в алфавитном порядке

if (typeid (i) .before (typeid (d)))

Если *typeid* не может создать объект класса *Type\_info,* оператор возбуждает исключительную ситуацию типа *Bad\_typeid.* Следовательно, при использовании *typeid* необходимо запрограммировать оператор *catch,* обрабатывающий эту ошибку.

***Указатели на функции-члены класса***

Можно ссылаться на функции-члены класса с помощью указателей, но не так, как на обычные функции. Функции-члены вызываются для объектов класса, и они получают скрытый указатель *this.* Поэтому адресация функций-членов требует нового подхода.

Указатель на функцию-член связан с именем ее класса. Например, для класса *A* можно объявить указатель на функцию-член в следующим образом:

double (А :: \* myp) (void);

Это означает, что указатель *myp* ссылается на функцию-член класса *А,* не имеет параметров и возвращает истинное значение двойной точности. Для объявления *myp* указателем, который ссылается на функцию, которая ничего не возвращает и имеет два целых параметры, можно записать:

void (А :: \* myp) (int, int);

В этих объявлениях не указывается конкретно, на какие функции-члены ссылается указатель *myp,* задается только вид функции, адрес которой может быть присвоена указателю. Остается создать объект класса и присвоить адрес его члена указателе, что соответствующий вид. К примеру:

class TFirstClass

{

private:

int count;

public:

TFirstClass ()

{

count = 0;

}

int Acs (void)

{

return count ++;

}

};

int (TFirstClass :: \* myfp) (void);

main ()

{

int i;

TFirstClass fc;

Cout << fc.Acs (); // Вызов функции-члена обычным средством

myfp = & TFirstClass :: Acs; // Адрес функции

cout << (fc. \* myfp) ();

TFirstClass \* fp = new TFirstClass;

cout << (fp -> \* myfp) ();

return 0;

}

Здесь *myfp* объявлен указателем на функцию-член класса *TFirstClass,* что не имеет аргументов и возвращает целое значение. Указатель может ссылаться на любые функции-члены класса, имеющие подобный вид. В функции *main ()* объявляется объект *fc* класса *TFirstClass.* Далее вызывается функция-член *Acs ()* класса *TFirstClass* обычным образом. Затем вызывается эта же функция с помощью указателя *myfp.* Для инициализации указателя ему присваивается адрес функции-члену *Acs ().* Можно заменить объявления указателя и присвоение одним оператором функции *main ():*

Int (TFirstClass :: \* myfp) (void) = & TfirstClass :: Acs ();

При вызове с указателем функции-члена необходимо придерживаться следующие правила:

• Обращаться к объекту класса.

• Взять в круглые скобки вызов функции.

Например, если *n* - целое переменная, и *fc* - объект класса *TFirstClass,* то запись

n = (fc. \* myp) ();

присвоит *n* результат функции *Acs ().* Двухсимвольного выражение *". \*"* Называется оператором ссылки на член и служит для размещения указателя на функцию-член объекта класса. Круглые скобки необходимы потому, что оператор вызова функции *()* имеет более высокий приоритет, чем *". \*".* ранее записанный оператор эквивалентен оператору

n = fc.Acs ();

Можно также вызвать функции-члены с помощью указателей, когда объекты адресуются другими переменными-указателями. В записанной программе сначала указателе *fp* присваивается адрес нового объекта класса *TFirstClass.* Затем в выражении

(fp -> \* myp) ();

в операторе вывода в поток вызывается функция-член *Acs ()* для объекта, адресованного указателем *fp.* Трьохсимвольний выражение -> \* - другой вид оператора ссылки на член. Круглые скобки опять необходимы. Оператор эквивалентный следующем:

fp -> Acs ();

В дополнение к адресации функций членов указателями, можно ссылаться на другие открытые члены класса. Например, если бы класс *TFirstClass* имел открытый член типа *double* на имя *balans,* можно было бы объявить указатель на этот член так:

double TFirstClass :: \* dataPtr;

Указатель *dataPtr* объявляется для адресации любых открытых данных-членов в классе *TFirstClass,* имеющих тип действительных значений двойной точности.

Для инициализации указателя надо использовать оператор:

dataPtr = & TFirstClass :: balance;

Или можно сказать указатель и присвоить ему адрес члена *balance:*

double TFirstClass :: \* dataPtr = & TFirstClass: balance;

В любом случае, указатель *dataPtr* теперь ссылается на член *balance* класса *TFirstClass.* На самом деле *dataPtr* не содержит адреса размещения в памяти, вместо этого в нем хранится смещение, по которому член *balance* размещается в объекте класса *TFirstClass.* Остается обратиться к адресованного члена через объект класса:

fc.balance = 1234.56;

cout << fc. \* dataPtr;

Запись *fc. \* DataPtr* аналогичный записи, использованному для вызова функции-члена. В нем используется оператор ссылки на член *".* *\*".* Здесь не нужны дополнительные круглые скобки, потому что выражение не содержит конфликтующих операторов.

***Статические члены класса***

Статические функции-члены обычно выполняют глобальные действия или инициализируют глобальные данные для всех объектов класса. Для объявления статической функции-члену необходимо отметить ключевое слово *static* перед объявлением функции-члена:

class TAnyClass

{

public:

static void Global (void);

};

Статические функции-члены не могут быть вируальнимы. реализуются они

так же, как и другие функции-члены:

void TAnyClass :: Global (void)

{

...

}

Статические функции-члены могут выполнять любые операторы, но они не получающие указателя *this* и не имеют доступа ни к каким данных членов или функций-членов классов. Для вызова статической функции-члена в программе используется не объект, а имя класса:

TAnyClass :: Global ();

Предполагается, что статическая функция-член выполняет действия, касающиеся всех объектов типа *TAnyClass.* В классах могут объявляться и статические данные- члены:

class TAnyClass

{

private:

static char c;

public:

char Get (void)

{

return c;

}

};

Здесь член *с* объявлен закрытым статическим членом класса. статические данные-члены также могут быть открытыми и защищенными. Существует только одна копия *TAnyClass :: c* независимо от того, сколько определяется объектов класса *TAnyClass* в программе. Статические данные-члены должны определяться и инициализироваться в программе с помощью глобальных объявлений:

char TAnyClass :: c = 'q';

На самом деле эта глобальная переменная доступна только функциям-членам

*TAnyClass.* Далее можно определить объявление класса *TAnyClass,* что

использует *c:*

TAnyClass x;

cout << x.Get ();

***Размещение объектов по заданным адресам***

Иногда может потребоваться разместить объект в заданный участок памяти, например в глобальный буфер в сегменте данных. Подобные объекты продолжают существовать и вне области их объявления и иногда их называют постоянными объектами.

Пример: размещение объекта по заданному адресу.

class TPersist

{

private:

int x, y;

public:

TPersist (int a, int b)

{

x = a;

y = b;

}

~ TPersist ()

{

x = 0;

y = 0;

}

void \* operator new (size\_t, void \* p)

{

return p;

}

friend ostream & operator << (ostream & os, TPersist & p)

};

char object [sizeof (TPersist)];

main ()

{

TPersist \* p = new (object) TPersist (10,20)

cout << \* p; // Буфер object:

cout << "Адрес:" << & object;

cout << "адрес \* p:" << & (\* p)

p-> TPersist :: ~ TPersist ();

return 0;

}

ostream & operator << (ostream & os, TPersist & p)

{

os << "x ==" << p. x << ", y ==" << p. y;

return os;

}

В классе *TPersist* оператор *new* перегружается необычным средством.

Вместо того, чтобы выделить память для указателя, функция перегрузки оператора просто возвращает указатель *p* типа *void.* В программе оператор *new* используется для хранения объекта класса *TPersist* в глобальном буфере.

Выражение *new (object)* вызывает перегружен оператор *new,* передавая ему адрес символьного буфера *object.* Поскольку перегружен оператор *new* просто возвращает переданный ему указатель *p,* этот оператор в действительности присваивает указателю *p* адрес объекта, а также вызывает конструктор класса *TPersist* для инициализации нового, выделенного в буфере, объекта.

В программе отражаются адреса символьного буфера *object* и объекта в буфере. Эти адреса идентичны, что свидетельствует о размещении объекта в указанном месте. Если запустить программу с отладчика и проследить за содержанием буфера *object* во время пошагового выполнения кода, то можно обнаружить, что в глобальном буфере появились значение 10 и 20. Это доказывает, что

перегружен оператор *new* не пользовался стандартными средствами выделение пространства в купе.

В программе приводится пример непосредственного вызова деструктора.

Этот прием используется для очистки объекта, созданного перегруженным оператором *new.* С ++ не распознается область действия объекта, сохраненного в буфере и не может вызвать деструктор класса автоматически. Для виртуальных деструкторов, даже в случаях с объектами производных

классов, можно использовать такую форму вызова деструктора:

p-> ~ TPersist ();

***Инициализация членов объекта класса***

Обычно данные члены объекта класса инициализируются в конструкторе. В качестве альтернативы можно инициализировать члены так, как если бы они имели конструкторы:

class A

{

private:

int x, y;

public:

A (): x (0), y (0) {}

};

В этой версии конструктора *A* членам *x* и *y* присваиваются нулевые значения. Инициализатор данных членов задаются к выполнению тела конструктора. Этот прием полезен для инициализации объектов других классов. Например, предположим, что класс *A* имеет член-объект класса *B* и конструктору

класса *B* необходим один целый параметр:

class B

{

int x;

public:

B (int n)

{

x = n;

}

};

Если в классе *A* объявлен член типа *B,* конструктор *A* обязан инициализировать этот объект. Поскольку конструктор не может быть вызван непосредственно, объект должен быть инициализирован с помощью альтернативного средства.

class A

{

private:

int x;

B z;

public:

A (int n): z (n)

{

x = n;

}

};

Член *z* - объект класса *B* Конструктор класса *A* объявлен с целым параметром *n,* который используется для инициализации объекта *z* перед выполнением тела конструктора. Если в классе есть несколько членов-объектов, можно инициализировать их, перечислив и разделив запятыми.

***Вложенные объявления в классе***

Внутри класса могут объявляться не только данные и функции, но и другие элементы: *typedef,* *struct* и другие классы. Такие вложенные объявления в классе бывают полезны, если они тесно связаны с классом. Они также позволяют двум или более классам объявлять элементы с одинаковым названием, что отличаются только именем соответствующих им классов.

Вложенный *typedef* надо использовать для экспорта типов данных с класса. Например, в классе *A typedef* используется для объявления C\_C для

int:

class A

{

public:

typedef int C\_C;

static C\_C ClassCount;

A ()

{

ClassCount ++;

}

~ A ()

{

ClassCount--;

}

};

В классе объявляется статическая переменная типа *C\_C* с именем *ClassCount.* Чтобы выделить память для этой переменной в программе необходимо сделать глобальное объявления:

A :: C\_C

A :: ClassCount;

Имя класса предшествует типа данных и имени переменной. Поскольку в классе *A* существует

*typedef C\_C,* в программе можно объявить переменную *ClassCount,* не зная действительного типа переменной. Без помощи со стороны программы класс может самостоятельно подсчитывать число созданных объектов его типа:

A \* cp1, \* cp2, \* cp3;

cp1 = new A;

cp2 = new A;

cp3 = new A;

cout << A :: ClassCount;

Последний оператор отобразит *3.*

Возможны сложные вложенные объявления. Внутри класса может объявляться структура:

class A

{

public:

struct CS

{

int x;

int y;

};

};

Затем в программе может быть объявлен объект типа *A ::* CS:

A :: CS k;

*k* можно использовать, как обычную структуру:

kx = 1;

ky = 2;

Также один класс может объявляться внутри другого:

class A

{

public:

class С

{

int x;

int y;

public:

С ()

{

x = 1;

y = 2;

}

int GetХ (void)

{

return x;

}

int GetУ (void)

{

return y;

}

};

};

Здесь в классе *A* объявляется внутренний класс *C.* В программе можно использовать этот вложенный класс так:

A :: C k;

cout << k.GetХ ();

cout << k.GetУ ();

Возможны и другие виды вложенных объявлений в классе, такие как перечисление констант. К использованию вложенных объявлений в классе применяется правило: ссылке на вложенный элемент должны предшествовать имя класса и оператор расширения области видения.