**2. ОСНОВНЫЕ СРЕДСТВА объектно-ориентированного программирования**

# лекция 5

**Тема лекции:** Простая наследственность. Объявления производных классов. Использование конструкторов и функций-членов производных классов. производных классов. Правила доступа к отдельным членам класса

# простая наследственность

Производный класс наследует из базового класса данные-члены и функции-члены, но не конструкторы и деструкторы.

Производный класс начинает свое существование с копирования членов базовых классов, включая все члены, унаследованные из более дальних родственных классов.

Рассмотрим базовый класс:

class TBase

{

private:

int count;

public:

TВase ()

{

count = 0;

}

void SetCount (int n)

{

count = n;

}

int GetCount (void)

{

return count;

}

};

Здесь count - закрытое данное-член. Конструктор по умолчанию TBase () инициализирует count нулевым значением. Функция-член SetCount () присваивает count новое значение. Функция-член GetCount () возвращает текущее значение count. Все эти функции являются встроенными. Создадим класс, имеющий все свойства TBase, но кроме того, способен увеличить значение объекта на заданный размер.

class TDerived: public TBase

{

public:

TDerived (): TBase () {}

void ChangeCount (int n)

{

SetCount (GetCount () + n)

}

};

Производный класс назван TDerived. Сразу по имени класса идет двоеточие и одно из ключевых слов - public, protected или private. После этих элементов следует имя базового класса, от которого производный класс получает наследство. В данном случае все открытые члены TBase становятся открытыми членами и в TDerived. Все закрытые члены остаются закрытыми в своем, сначала объявленном классе. В TDerived нельзя получить доступ к этим членам TBase.

Можно объявить базовый класс закрытым:

class TDerived: private TBase {};

В этом случае все открытые члены TBase становятся закрытыми членами в TDerived. Третий тип спецификатор protected рассматривается позже.

Обычно в производном классе есть конструктор, если он и в базовом классе. Кроме того, конструктор производного класса должен вызвать конструктор базового класса. В TDerived в строке

TDerived (): TBase () {}

объявляется конструктор производного класса TDerived () и вызывается конструктор базового класса с помощью специального обозначения : TBase (). Нельзя вызывать конструкторы базовых классов в операторах. В конструкторе могут выполняться операторы. Например, конструктор TDerived () можно объявить встроенным следующим образом:

TDerived (): TBase ()

{

cout << "Инициализация \ n";

}

Конструкторы производных классов не обязательно должны быть встроены. К примеру:

class TDerived: public TBase

{

public:

TDerived ();

void ChangeCount (int n)

{

SetCount (GetCount () + n)

}

};

В этом случае необходимо реализовать конструктор TDerived () отдельно:

TDerived :: TDerived (): TBase ()

{/ \* Операторы конструктора \* /}

Производный класс наследует сount с базового класса. В классе TDerived нельзя получить к классу TВase непосредственный доступ, поскольку count закрыт в TBase. Только члены класса могут иметь доступ к закрытым членам этого же класса. В производном классе можно объявить свои собственные закрытые члены:

class TDerived: public TBase

{

private:

int SecondCount;

void ChangeCount (int n)

{....}

};

TDerived наследует свойства TBase, включая закрытый член базового класса count. В новом классе добавился новый закрытый член SecondCount. Для его инициализации реализуем конструктор:

TDerived :: TDerived (): TBase

{

SecondCount = 0

}

Вызов конструктора TBase () инициализирует объект класса TBase. оператор присваивания

SecondCount = 0

в конструкторе производного класса инициализирует объект класса TDerived. Конструктор базового класса должен вызываться первым. В новом классе TDerived объявлена ​​открытая функция-член ChangeCount () для увеличения значения члена count, унаследованного из класса TBase. В функции ChangeCount () нельзя получить непосредственный досуп к члену count, так как он закрыт в классе TBase:

void ChangeCount (int n)

{

сount + = n;

}

Этот оператор вызывает сообщение об ошибке при компиляции.

# Правила использования открытых, закрытых и защищенных членов класса

Защищенный член класса - нечто среднее между закрытым и открытым членом. Подобно закрытым членам, защищенные члены доступны только функциям-членам класса. Вне класса защищенные члены невидимы. Подобно открытых членов, защищены наследуются производными классами и доступны функциям-членам производных классов.

• Закрытые члены класса доступны только в классе, где они объявлены.

• Защищенные члены доступны членам их собственного класса и всем членам производных классов.

• Открытые члены доступны членам их собственного класса, членам производных классов и всех других пользователей этого класса.

• По умолчанию члены класса, наследуемого закрыты. Спецификаторы private, protected и public влияют на статус унаследованных членов.

• Члены открытого базового класса в производном классе сохраняют свои спецификаторы доступа.

• Открытые члены защищенного базового класса становятся защищенными в производном классе. Защищенные и закрытые члены сохраняют свои первоначальные спецификации доступа.

• Все члены закрытого базового класса становятся закрытыми в производном классе независимо от начальной спецификации доступа этих членов.

Третий случай наиболее существенно влияет на такие члены:

class TBase

{

protected:

int x;

public:

int y;

.......

};

class TDerived: private TBase

{

public:

void f (void);

};

class TDesc: public TDerived

{

public:

void g (void);

};

Функция-член f () класса TDerived имеет доступ к членам x и y, что унаследованные от класса TBase. Но поскольку TBase объявлен закрытым для класса TDeruved, то статус членов x и y меняется на private. Функция-член g () класса TDesc, производного класса от TDerived, не имеет доступа к членам x и y, несмотря на то, что сначала эти члены имели статус protected и public.

Спецификатор базового класса потенциально влияет на все члены, наследуются. При необходимости сделать закрытыми в производном классе только некоторые из унаследованных членов, можно выборочно квалифицировать один или несколько из них. пример:

class A

{

public:

int x;

A ()

{

x = 0;

}

void Display (void)

{

cout << "Конструктор В \ n";

}

};

запись:

B object;

object.Display ();

ложный, так как класс А объявлен закрытым для класса В. Чтобы сделать функцию-член класса А Display () открытой для класса, оставив данное-член класса А х закрытым, запишем:

class B: private A

{

public:

A :: Display ();

B (): A ()

{

cout << "Конструктор В \ n";

}

};

Теперь можно создать объект типа В и вызвать функцию Display () из класса А.

# Использование конструкторов и деструкторов

В производном классе деструктор нужен только при наличии членов, которые необходимо удалять, когда объект производного класса выходит из области видения. Рассмотрим базовый класс:

class TBase

{

private:

char \* basep;

public:

TBase (count char \* s)

{

basep = strdup (s)

}

~ TBase ()

{

delete basep;

}

const char \* GetStr (void)

{

return basep;

}

};

Класс TBase имеет один защищенный член - указатель basep на строку символов. Конструктор класса вызывает библиотечную функцию strdup () для создания копии строчной аргумента в виде динамической переменной с привласнeнням выделенной адреса указателе basep. Деструктор ~ TBase () освобождает этот блок памяти, когда объект типа TBase выходит из области видения. Объявим производный класс:

class TDerived: public TBase

{

private:

char \* uppercasep;

public:

TDerived (const char \* s);

~ TDerived ()

{

delete uppercasep;

}

const char \* GetUStr (void)

{

return uppercasep;

}

};

К членам, унаследованных с TBase, TDerived добавляет закрыт член - символьный указатель с именем uppercasep, который ссылается на копию строки, хранящейся в TBase, с символами, превращенными в большие. Деструктор производного класса удаляет этот строку. Конструктор TDerived () можно реализовать следующим образом:

TDerived :: TDerived (const char \* s): TBase (s)

{

uppercasep = strupr (strdup (s));

}

Конструктор TDerived () вызывает конструктор TBase (), который копирует строку, на который ссылается s, до кучи и указатель, инициализирует basep адресу строки. После завершения этой операции конструктор производного класса создает еще одну копию строки, превращая его символы на прописные буквы с помощью библиотечной функции strupr () и присваивает uppercasep адрес этой копии строки. При создании объекта типа TDerived создается 2 копии строки: одна неизменная и одна, символы которой преобразованы в прописные.

Создадим такой объект:

TDerived country ( "Ukrainе")

Представление этого объекта в памяти следующее:

начальная строка, инициализирован конструктором базового класса и копия строки из преобразованных строчных букв на прописные, инициализирована конструктором производного класса:

Объект coutnty класса TBase:

char \* basep> Ukrainе

char \* uppercasep -> UKRAIN

Объекта country принадлежит два блока памяти. Когда объект выходит из области видение, его деструктор освобождает эту память и возвращает ее купе для последующего использования. Деструктор класса TBase освобождает память, на которую ссылается член basep. Деструктор класса TDerive освобождает память, на которую ссылается uppercasep. В отличие от конструкторов, деструктор производного класса никогда не вызывает непосредственно деструктор базового класса. С ++ автоматически сначала вызывает деструктор производного класса, потом - деструктор базового класса. Порядок вызова деструкторов - обратный порядок вызова конструкторов.

В производном классе можно изменить поведение унаследованного элемента. Например, можно объявить переменную или функцию с тем же именем, что и в базовом классе, но которая имеет другой тип или выполняет другие действия. Это называется переопределением. Если в производном классе есть элемент с таким же именем, что и в базовом, используется элемент производного класса. Если появляется необходимость вызвать элемент базового класса, нужно применить оператор расширения области видения.

При создании новой программы желательно поискать среди классов такие, которые могут потребоваться в дальнейшем. Использовав их в качестве базовых, можно вывести новые классы, то есть использовать повторно уже существующий код.