**лекция 6**

**Тема лекции**: Виртуальные функции. Пример использования виртуальных функций. Использование конструкторов и деструкторiв базовых и производных классов. «Чистые» виртуальные функции. Абстрактные базовые классы. Виртуальные деструкторы. Множественная спадкоемнiсть. Использование виртуальных базовых классов

**Виртуальные функции**

Виртуальная функция - это функция, вызов которой зависит от типа объекта. В классическом программировании необходимо было задавать тип при написании кода. В объектно-ориентированном программировании можно писать виртуальные функции так, чтобы объект определял, какую функцию необходимо вызывать, во время выполнения программы. То есть с помощью виртуальных функций объект определяет свои собственные действия. Технику использования виртуальных функций называют еще полиморфизмом.

Согласно правилам С++ указатель на базовый класс может ссылаться на объект этого класса или на объект любого другого класса, производного от базового.

Это правило - ключ к пониманию того, как используются виртуальные функции. Рассмотрим ряд следующих классов: А, В, С, где класс В выводится по классу А, а С - с В. В программе объекты классов А, В, С могут объявляться с помощью таких операторов:

A aObject;

B bObject;

C cObject;

По определению указатель типа А может ссылаться на любой из трех объектов, поскольку они связаны отношением наследования.

К примеру:

A \* p;

p = & cObject;

Здесь присваивается адрес объекта cObject указателю на базовый класс. Несмотря на то, что указатель р имеет тип А \*, а не С \*, он может ссылаться на объект типа C, потому что класс С является производным от класса А. Эта зависимость объектов и указателей работает только в одном направлении.

Этот принцип становится особенно важным, когда в связанных родственными отношениями классах определяются виртуальные функции. Функции имеют такой же вид и программируются так же, как и обычные функции, только их объявления опережается ключевым словом virtual. Например, класс А может объявить виртуальную функцию vf ():

class A

{

public:

virtual void vf ();

};

Виртуальная функция может объявляться с параметрами, она может возвращать значение, как и любая другая функция. В классе может объявляться столько виртуальных функций, сколько потребуется. Они могут быть открытыми, защищенными и закрытыми членами класса. В классе В, производном от того, можно объявить виртуальную функцию с тем же именем. Это - функция, замещает. Запишем оператор, вызывает виртуальную функцию vf () для объекта, на который ссылается г:

p -> vf ();

Указатель г. может хранить адрес объекта типа А, В или С. Следовательно, при выполнении программы этот оператор вызывает виртуальную функцию класса объекта, на который ссылается г. Тот самый оператор вызывает функцию класса текущего объекта - действие, определенная во время выполнения.

Допустим, создается графическая программа и разрабатываются различные классы для таких фигур, как квадраты, круги, линии. Все эти классы взаимосвязаны (все они - фигуры), поэтому можно сначала создать базовый класс Shape:

class Shape

{

public:

virtual void Draw ();

};

Еще неизвестно, какие типы фигур потребуются, возможно, будут объявлены новые фигуры. Необходимо где-то хранить коллекцию объектов Shape, пожалуй, в массиве указателей. Пусть он объявлен глобальным и, следовательно, все указатели инициализированы нулевым значением.

Shape \* picture [100];

Каждый элемент массива - указатель типа Shape \*, который может ссылаться на объект типа Shape или на объект любого производного от него класса. Поскольку в классе Shape объявлена ​​виртуальная функция Draw (), можно запрограммировать цикл, вызывает функцию Draw () для указателей из массива picture.

int i = 0;

while (i <100 && picture! = 0)

{

picture [i] -> Draw ();

i ++;

}

Итак, написанный код, который рисует картинку до того, как появилась хотя бы одна существующая фигура. Этот код можно скомпилировать и сохранить в библиотеке или в .ОBJ файле. Позже можно вывести новые классы из класса Shape, сохранить объекты этих классов в массиве picture и цикл будет вызывать виртуальные функции Draw () этих объектов для рисования соответствующих изображений. Этот код не требует точного задания типов данных объектов, на которые ссылаются указатели picture, нужно только, чтобы эти объекты были производными от Shape. Объекты определяют при исполнении, какую функцию Draw () надо вызывать. Выведем классы Circle и Line из класса Shape.

class Circle: public Shape

{

public:

virtual void Draw ();

};

class Line: public Shape

{

public:

virtual void Draw ();

};

Теперь нужно запрограммировать функции, замещающих Draw () в классах Circle и Line для рисования круга и полосы. При выполнении программы цикл, рисует, вызывает необходимые виртуальные функции, исходя из тела объекта, который адресуется. В массив picture [] можно добавлять фигуры так:

picture [0] = new Circle;

picture [1] = new Line;

picture [2] = new Triangle;

Невозможно угадать, какие виды объектов будет содержать массив picture [], но это не имеет значения для цикла рисования. Этот цикл способен нарисовать любой объект класса, производного от класса Shape. Вызовы виртуальных функций-членов компонуются во время выполнения программы с помощью техники, называется поздним связыванием. Вызовы обычных функций-членов компонуются во время компиляции (раннее связывание). В технической реализации, адреса виртуальных функций-членов хранятся в виртуальной таблице. Поиски нужного адреса отнимают время и влияют на быстродействие программы. Виртуальные функции-члены наследуются производными классами, как и обычные функции-члены. В последнем из производных классов в линии наследования, можно удалить ключевое слово virtual.

**Чистые виртуальные функции**

Чистая виртуальная функция - пустое место, что ожидает своего заполнения производным классом. Объявления чистой виртуальной функции в классе завершается = 0. Например:

class AbstractClass

{

public:

virtual void f1 (void);

virtual void f2 (void) = 0;

};

f2 () - чистая виртуальная функция-член класса. Компилятор не потребуется реализации функции-члена f2 (), в отличие от других виртуальных функций-членов, объявленных в классе.

Если класс содержит хотя бы одну чистую виртуальную функцию, он называется абстрактным классом. Абстрактный класс - это схема, на основании которой создаются производные классы. В С ++ нельзя создавать объекты, имеющие типы абстрактных классов. Например, компиляция такой строки:

AbstractClass myObject;

выдаст сообщение об ошибке "Can not create an instance of class AbstractClass" - "Не могу создать объект класса AbstractClass".

Другие функции-члены этого класса могут вызвать чистую функцию-член.

void AbstractClass :: f1 (void)

{

...

 f2 ();

...

}

Чтобы использовать абстрактный класс, надо вывести из него новый класс:

class MyClass: public AbstractClass

{

public:

virtual void f2 (void);

}

Производный класс MyClass наследует чистую виртуальную функцию-член, но объявляет ее без знаков "= 0". В другом месте программы нужно реализовать f2 ():

void MyClass :: f2 (void)

{

...

}

Обращение к исходной чистой виртуальной функции-члена теперь переадресованы MyClass :: f2 (). С помощью объявлений чистых виртуальных функций-членов в производном классе можно вставлять заглушки к коду, который может быть вызван другими функциями-членами. Теперь, когда все виртуальные функции-члены реализованы, компилятор сможет воспринять объекты типа MyClass:

MyClass myObject;

myObject.f1 ();

Второй из операторов приведет к вызову унаследованной f1 (), что вызывает f2 () с MyClass.

Чистые виртуальные функции могут быть открытыми, но чаще всего они защищены, поскольку предполагается, что функция будет реализована в производном классе.

**Виртуальные деструкторы**

Виртуальные деструкторы обычно применяются в случаях, когда в некотором классе необходимо удалить объекты производного класса, на которые ссылаются указатели на базовый класс. Рассмотрим класс, запоминает строковое значение:

class TBase

{

private:

char \* sp1;

public:

TBase (const char \* s)

{

sp1 = strdup (s)

}

virtual ~ TBase ()

{

delete sp1;

}

};

Конструктор класса TBase выделяет пространство для строки, обращаясь к функции strdup () и сохраняет адрес строки в указателе sp1. Виртуальный деструктор освобождает эту область, когда объект типа TBase выходит из области видения.

Выделяем новый класс по TBase:

class TDerived: public TBase

{

private:

char \* sp2;

public:

TDerived (const char \* s1, const char \* s2): TBase (s1)

{

sp2 = strdup (s2)

}

virtual ~ TDerived ()

{

delete s2;

}

};

Новый класс сохраняет строку, на который ссылается s2. Новый конструктор вызывает TBase (), передавая строку к базовому классу, а также выделяет память для второй копии строки удаляется деструктором класса.

Когда объект TDerived выходит из области действия, важно, чтобы обе копии строки были удалены. Предположим, что указателю на класс TBase присвоен адрес объекта TDerived:

TBase \* pBase;

pBase = new TDerived ( "String1", "String2")

...

delete pBase;

При освобождении памяти компилятор вызывает нужен деструктор. В данном случае указатель pBase ссылается на объект производного класса, то есть вызывается деструктор класса TDerived. Объекты сами определяют, какой деструктор вызвать, потому что деструкторы объявлены виртуальными. Если деструкторы не виртуальны, вызовется деструктор базового класса, оставив вторую копию строки в памяти.

Конструкторы вызываются только при создании объекта, а тип объекта, создаваемого должен быть известен компилятору. Поэтому конструктор не может быть виртуальным.

Конструкторы не бывают виртуальными, но могут вызвать виртуальную функцию. А для разрушения объекта тип не важен, поэтому возможны виртуальные деструкторы.

**Множественная преемственность. Виртуальные базовые классы**

При множественной преемственности производный класс может иметь несколько базовых классов.

Для вывода нового класса из нескольких базовых надо перечислить имена базовых классов после имени нового:

class D: public A, public B, public C

{

   ....

};

Спецификаторы доступа можно использовать различные и в произвольном порядке.

Производный класс наследует все свойства всех базовых классов. Если в двух базовых классах объявляется функция-член с тем же именем, то для доступа к ним используется оператор расширения области видения.

Если классы А, В, С имеют конструкторы по умолчанию, то в производном классе D их можно вызвать так:

class D: public A, public B, public C

{

public:

D (): A (), B (), C () {}

};

Конструктор можно реализовать отдельно, в другом месте программы:

D :: D (): A (), B (), C ()

{

....

}

Конструкторы вызываются в том порядке, в котором объявлены их базовые классы.

В прямом ациклические графе (ПАГ) наследование члены класса могут появляться неоднократно. Рассмотрим ПАГ множественной преемственности:

Рarent

Сhild1 Сhild2

GrandChild

Рис. 2.1 Прямой ациклический граф множественной преемственности

Элементы класса Parent появляются дважды в классе GrandChild. Первый набор наследуется через Child1, второй набор - через Child2. Такое наследование бывает нежелательным. Виртуальные базовые классы обеспечивают механизм для предотвращения дублирования элементов в классе.

class Parent {};

class Child1: public virtual Parent {....};

class Child2: public virtual Parent {....};

class GrandChild: public Child1, public Child2 {....};

Здесь класс GrandChild многократно наследует поля класса Parent. Благодаря использованию виртуальных базовых классов, объекты класса GrandChild имеют по одному полю данных из базового класса. Виртуальные базовые классы инициализируются перед любыми не виртуальными базовыми классами и в том порядке, в котором они объявлены. Если виртуальный базовый класс имеет хотя бы один конструктор, то он должен иметь конструктор по умолчанию. С производного класса нельзя прямо вызвать параметризованный конструктор виртуального базового класса.