Лекция 5. Виртуальные функции

**Взаимосвязь объектов производного и базового классов**

Объект производного класса хранит в памяти объекты базового класса. На этапе выполнения программы возможно выделение объекта базового класса из объекта производного класса, а также выделение объекта производного класса по объекту его базового класса. Для этого используется **косвенное обращение к элементам класса.** Указатели на объекты порождённых классов совместимы по типу с указателями на объекты базового класса. Поэтому указателю на объект базового класса можно присваивать значение указателя на объект производного класса без явного преобразования типов. Присваивание указателю на объект производного класса значения указателя на объект базового класса неявным образом запрещено (разрешено явное приведение типов).

Пример. Организация доступа к методам посредством указателей.

class BCls

{

public:

void outs(){

cout<<"Base class"<<endl;

}

};

class PCls01:public BCls

{

public:

void outs(){

cout<<"Class 01"<<endl;

}

};

class PCls02:public BCls

{

public:

void outs(){

cout<<"Class 02"<<endl;

}

};

int main()

{

BCls b, \*ukb;

PCls01 x, \*ukp01;

PCls02 y, \*ukp02;

ukb=&x; ukb->outs();//base class

ukb=&y; ukb->outs();//base class

ukb=&b; ukb->outs();//base class

//ukp01=ukb;//Ошибка

ukp01=(PCls01 \*)ukb; ukp01->outs();//Class 01

ukp01=&x; ukp01->outs();//Class 01

ukp02=(PCls02 \*)ukb; ukp02->outs();//Class 02

ukp02=&y; ukp02->outs();//Class 02

return 0;

}

При работе:

1. Компилятор оценивает не содержимое, а тип указателя.

2. По адресу объекта производного класса можно получить адрес любого объекта базового класса.

3. По адресу объекта базового класса, используя операцию явного приведения типа, можно получить адрес любого объекта производного класса.

**Виртуальные функции**

**Виртуальный метод** - это метод класса, который может быть переопределён в производных классах таким образом, что конкретная реализация метода будет определяться во время выполнения программы.

Виртуальные методы позволяют создавать общий код программы, который может работать как с объектами базового класса, так и с объектами любого класса-потомка. Базовый класс определяет способ работы с объектами, а производные классы предоставляют реализацию этого способа.

\*Для прямого доступа к производным методам через указатель необходимо сделать метод в базовом классе виртуальным.

В предыдущем примере исправление: добавим в заголовок в базовом классе virtual void outs().

Вывод:

Class 01

Class 02

base class

base class

Class 01

base class

Class 02

В случае использования виртуальных функций компилятор выбирает метод не в соответствии с типом, а в соответствии с полученным адресом. Если базовый класс содержит виртуальную функцию, то одноимённая функция производного класса будет вызываться для объектов своего класса даже в случае использования указателя на базовый класс. Виртуальные функции вызываются только для объектов, принадлежащих определённому классу. Нельзя объявлять глобальную или статическую функцию виртуальной. Функция, объявленная в производном классе, переопределяет виртуальную функцию в базовом классе только в случае совпадения имени функции, количества, типов и порядка следования параметров.

Пример. Применение виртуальных функций для классификации объектов, помещаемых в контейнер.

class BCls{

public:

virtual void vivod(){

cout<<"Base class"<<endl;

}

};

class PCls01:public BCls{

public:

void vivod(){

cout<<"Class 01"<<endl;

}

};

class PCls02:public BCls{

public:

void vivod(){

cout<<"Class 02"<<endl;

}

};

class PCls03:public BCls{

public:

void vivod(){

cout<<"Class 03"<<endl;

}

};

void main(){

BCls \*mas[4];

mas[0]=new BCls;

mas[1]=new PCls01;

mas[2]=new PCls02;

mas[3]=new PCls03;

print\_mas(mas, 4);

//+Очистить память

}

Базовые классы и их виртуальные функции могут быть созданы и откомпилированы до определения производных классов.

Пример. Применение виртуальных функций.

class BCls{

public:

int k;

BCls(int a):k(a){}

virtual void vivod(BCls &x){

cout<<"Base class:k="<<x.k<<endl;

}

void operator ++(){

++(this->k);

vivod(\*this);

}

};

class PCls:public BCls{

public:

PCls(int a):BCls(a){}

void vivod (BCls &x){

cout<<"Derivative class: k= "<<x.k<<endl;

}

};

int main()

{

PCls x(10);

x.vivod(x);

++x;

return 0;

}

Для подавления виртуального обращения к методу можно использовать операцию разрешения области действия (::). Такое обращение позволяет избежать потерь ресурсов на динамическое определение в задачах, критичных к быстродействию.

Пример. Сформировать список товаров. Список должен содержать информацию о наименовании и цене товаров. Для акционных товаров должен указываться размер скидки.

class Ctovar

{

char \*product;

double cena;

public:

Ctovar(char \*a, double b):cena(b)

{

product=new char[strlen(a)+1];

strcpy(product, a);

}

~Ctovar(){delete []product;}

virtual void vivod(){

cout<<product<<":"<<cena;

}

};

class Cskidka: public Ctovar{

int discount;

public:

Cskidka(char \*a, double b, int c):Ctovar(a, b), discount(c){}

void vivod(){

Ctovar::vivod();

cout<<"-"<<discount<<"%";

}

};

struct tstk{

tstk \*a;

void \*inf;

void add(tstk \*\*sp, void \*inform)

{

tstk \*spt=new tstk;

spt->a=\*sp;

spt->inf=inform;

\*sp=spt;

}

};

class Cshop{

tstk \*goods;

public:

Cshop():goods(NULL){}

void operator+=(void \*x)

{

goods->add(&goods, x);

}

void operator()(){

cout<<"List"<<endl;

for(tstk \*x=goods;x!=NULL;x=x->a)

{

((Ctovar\*)x->inf)->vivod();

cout<<endl;

}

}

~Cshop()

{

tstk \*spt;

while(goods!=NULL){

spt=goods;

goods=goods->a;

delete spt;

}

}

};

int main()

{

Cshop tov;

tov+=new Ctovar("Fridge", 432.3);

tov+=new Cskidka("Table", 126.3, 10);

tov+=new Cskidka("TV", 692.3, 15);

tov+=new Ctovar("Chair", 63.4);

tov();

return 0;

}

**Виртуальные деструкторы**

Пример. Работа деструктора в объекте производного класса.

class XY

{

double x, y;

public:

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

~XY() {cout<<"Delete XY"<<endl;}

};

class Z:publicXY

{

char Z;

public:

Z():XY(),Z(0){}

~Z() {cout<<"Delete Z"<<endl;}

};

void main()

{

XY \*ukb=new Z;

delete ukb;

}

Происходит утечка памяти, так как деструктор для производного класса не выполняется. Если в примере деструктор сделать виртуальным, то утечки памяти не будет:

virtual ~XY() {cout<<"Delete XY"<<endl:}

**Абстрактные классы и чисто (строго) виртуальные функции**

virtual тип <имя\_функции> (<список параметров>)=0;

Производный класс может не содержать переопределения виртуальной функции. В этом случае будет доступна с учётом прав доступа виртуальная функция базового класса.

Для запрета использования виртуальной функции базового класса используются чисто виртуальные функции. Функция, являющаяся чисто виртуальной, должна быть переопределена в производном классе.

Любой класс, содержащий хотя бы одну чисто виртуальную функцию, называется **абстрактным**. Определение объектов абстрактного класса запрещено. Он предназначен только для использования в качестве базового класса. Если в производном классе отсутствует переопределение виртуальной функции, то он тоже становится абстрактным.

Особенности абстрактных классов:

1) Нельзя объявлять объекты абстрактного класса.

2) Нельзя использовать имя абстрактного класса в качестве типа аргумента функции и в качестве типа возвращаемого значения.

3) Запрещено явное преобразование к типу абстрактного класса.

4) Разрешено использование указателя или ссылки на абстрактный класс.

Пример.

class ACls

{

protected:

int x;

public:

AСls(int a):x(a) {}

virtual void vivod()=0;

};

class PCls01:public AСls

{

protected:

int y;

public:

PCls01(int a, int b):ACls(a), y(b) {}

void viv()

{cout<<x<<" "<<y<<endl;}

};

class PCls02:AСls

{

int z;

public:

PCls02(int a, int b):ACls(a), z(b) {}

void vivod()

{cout<<x<<" "<<z<<endl;}

};

class PCls03:public PСls01

{

int w;

public:

PCls03(int a, int b, int c):PCls01(a, b), w(c) {}

void vivod()

{cout<<x<<" "<<y<<" "<<w<<endl;}

};

int main(){

//ACls ob01(1);//Ошибка - невозможно создать экземпляр абстрактного класса

//PCls01 ob02(2, 3);//Ошибка - невозможно создать экземпляр абстрактного класса

PCls02 ob03(4, 5);

PCls03 ob04(5, 7, 8);

ob03.vivod();

ob04.vivod();

return 0;}

Пример. Демонстрация эволюционного процесса создания программы.

class BCls

{

protected:

int x;

public:

BCls(int a=0):x(a) {}

virtual void vivod()=0;

};

class PCls01:public BCls

{

int y;

public:

PCls01(int a, int b=0):BCls(a), y(b) {}

virtual void vivod()

{cout<<"Class 01"<<x<<" "<<y;}

};

class PCls02:public BCls

{

int y;

public:

PCls02(int b=0): y(b) {}

virtual void vivod()

{cout<<"Class 02"<<x<<" "<<y;}

};

class Ccount{

int k;

public:

Ccount(int a=0):k(a) {}

void operator<<(BCls&m)

{

cout<<endl<<++k<<")";

m.vivod();

}

};

int main(){

Ccount out;

out<<PCls01(2,10);

out<<PCls02(20);

out<<PCls01(5, 30);

out<<PCls02(40);

cout<<endl;

return 0;

}