Лекция 4. Наследование

**Производный класс**

**Наследование** - механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменён в новом классе. Существующий класс называется **базовым**, а новые классы - **производными**. Производный класс наследует описание базового класса, которое можно изменить добавлением новых и модификацией существующих элементов, а также путём изменения прав доступа к элементам. Наследуемые компоненты не перемещаются в производный класс, а остаются в базовом классе. Сообщения, обработку которых не могут выполнить методы производного класса, передаются на обработку в базовый класс. Поиск полей, отсутствующих в производном классе, будет вестись в базовом классе. Поля и методы базового класса могут быть переопределены в производном классе. В этом случае соответствующие элементы базового класса становятся недоступны в производном классе. Для доступа к методам в базовом классе используется операция разрешения области видимости. Допускается множественное наследование - использование нескольких не связанных классов для создания производного класса.

class имя\_класса: список\_базовых\_классов

{

Описание\_полей\_и\_методов\_класса

}

Пример. Создать производный класс.

#include <iostream>

using namespace std;

class Cls{

protected:

double s;

int p;

public:

Cls(double a, int b):s(a), p(b)

{}

void get\_sp()

{

cout<<"s="<<s<<"p="<<p<<endl;

}

void operator++()//С параметрами будет постфиксная форма

{s++; p++;}

};

class PCls:public Cls

{

public:

PCls(double a, int b):Cls(a, b){}

void operator--()

{s--; p--;}

};

void main()

{

Cls x(2.4, 3);

PCls y(2.4, 3);

cout<<"x: "; x.get\_sp();

cout<<"y: "; y.get\_sp();

--y;

cout<<"y: "; y.get\_sp();

}

По умолчанию в производном классе унаследованные компоненты получают спецификатор доступа private. Явно изменить спецификатор доступа при наследовании можно, указав новый спецификатор перед именем базового класса. Атрибуты доступа в производном классе изменяются следующим образом:

|  |  |  |
| --- | --- | --- |
| Атрибут, указанный при наследовании | Атрибут в базовом классе | Атрибут, полученный в производном классе |
| public | public | public |
| protected | protected |
| private | недоступен |
| protected | public | protected |
| protected | protected |
| private | недоступен |
| private | public | private |
| protected | private |
| private | недоступен |

Пример. Создать базовый и производный классы с различными атрибутами к полям.

class Cls{

private:

int a;

protected:

int b;

public:

int c;

};

class PCls:public Cls

{

private:

int d;

protected:

int f;

public:

Cls::b;

int g;

};

int main()

{

Cls x;

x.a;//Доступ запрещён - ошибка

x.b;//Доступ запрещён - ошибка

x.c;//Доступ разрешён

PCls y;

y.a;//Доступ запрещён - ошибка

y.b;//Доступ разрешён

y.c;//Доступ разрешён

y.d;//Доступ запрещён - ошибка

y.f;//Доступ запрещён - ошибка

y.g;//Доступ разрешён

return 0;}

Пример. Создать базовый и производный классы, использующие собственные методы для доступа к полям.

class Cls

{

private:

int a;

protected:

int b;

public:

int c;

};

class PCls:public Cls//Возможен доступ ко всем полям

{

public:

/\* int read\_a(){

return a;//Доступ запрещён - синтаксическая ошибка

}\*/

int read\_b()

{return b;}//Доступ разрешён

int read\_c()

{return c;}

};

int main(){

PCls x;

x.read\_b();

x.read\_c();

return 0;

}

**Конструкторы производных классов**

Как правило, конструкторы объектов производных классов вызывают конструкторы базовых классов. При описании конструктора производного класса после : записывается в произвольном порядке вызов конструкторов базовых классов.

Пример. Создать базовый и производный классы, использующие собственные методы для доступа к полям.

class Cls01

{

int x;

public:

Cls01(int a):x(a) {}

void outx(){

cout<<x<<endl;

}

};

class Cls02

{

char st[10];

public:

Cls02(char s[]) {strcpy(st, s);}

void outst(){

cout<<st<<endl;

}

};

class PCls:public Cls01, public Cls02//Возможен доступ ко всем полям

{

public:

PCls(int xx, char ss[]):Cls01(xx), Cls02(ss)

{}

};

int main(){

PCls x(22, "aaa");

x.outx();//Выводит 22

x.outst();//aaa

return 0;

}

**Перегрузка методов**

Если методы базового класса заменяются одноименными методами производного класса, то имеет место перегрузка методов. В этом случае при использовании объектов базового класса вызывается метод базового класса, а объекты производного класса вызывают соответствующие методы производного класса.

Пример. Создать базовый и производный классы для расчета суммы полей.

const int n=5;

class Cls01

{

protected:

int x;

public:

Cls01(int a):x(a){}

void outsum()

{

cout<<"Sum="<<x<<endl;

}

};

class Cls02

{

protected:

int y,z;

public:

Cls02(int a, int b):y(a), z(b){}

void outsum()

{

cout<<"sum="<<y+z<<endl;

}

};

class PCls:public Cls01, public Cls02

{

int mas[n];

public:

PCls(int a, int b, int c, int p):Cls01(a), Cls02(b,c)

{

for(int i=0; i<n; i++) mas [i]=p\*i;

}

void outsum()

{

int s=x+y+z;

for(int i=0; i<n; i++) s+=mas[i];

cout<<"Sum="<<s<<endl;

}

};

int main()

{

Cls01 g=5;

g.outsum();//5

Cls02 r(4, 6);

r.outsum()//10

PCls w(2, 4, 6, 1);

w.outsum();//22

return 0;

}

**Множественное наследование** - это одновременное наследование от нескольких не связанных базовых классов.

Пример. Ввести и вывести на экран информацию о работниках предприятия, среди которых могут быть как штатные сотрудники, так и студенты, проходящие практику.

class Cfio

{

char fio[20];

public:

void vvod()

{

cout<<"Enter the name"<<endl;

gets(fio);

}

void vivod()

{

cout<<"Name: "<<fio<<endl;

}

};

class Cotd

{

int notd;//Номер отдела

public:

void vvod()

{

cout<<"Enter your department number"<<endl;

cin>>notd;

}

void vivod()

{

cout<<"Department: "<<notd<<endl;

}

};

class Csotr:public Cfio, public Cotd

{

char dolg[20];

public:

void vv()

{

Cfio::vvod();

Cotd::vvod();

cout<<"Enter job title"<<endl;

gets(dolg);

}

void viv()

{

Cfio::vvod();

Cotd::vvod();

cout<<"Job title: "<<dolg<<endl;

}

};

class Cstud:public Cfio, public Cotd

{

int ngr;

public:

void vv()

{

Cfio::vvod();

Cotd::vvod();

cout<<"Enter group number"<<endl;

cin>>ngr;

}

void viv()

{

Cfio::vvod();

Cotd::vvod();

cout<<"Group number: "<<ngr<<endl;

}

};

int main()

{

Cstud stud;

Csotr sotr;

stud.vv();

stud.viv();

//stud.vivod();//Ошибка - неоднозначный уровень доступа "vivod"

stud.Cotd::vivod();

return 0;

}

A

C

B

D

Неопределенность в виде ромба может возникнуть, если класс является производным от двух базовых классов, которые в свою очередь являются производными одного базового класса. В этом случае при обращении к переменным базового класса компилятор сообщает об ошибке, что вызвано тем, что компилятор обрабатывает каждый путь наследования отдельно. В результате объект D будет содержать два разных объекта A. Поэтому при использовании таких объектов требуется указать путь наследования.

Пример.

class A

{

public:

int k;

};

class B:public A

{};

class C:public A

{};

class D:public B, public C

{};

…

D y;

//y.k//Ошибка - неоднозначный уровень доступа "k"

y.B::k;

**Включение классов в классы**

Включение, не являясь наследованием, может выполнять аналогичные функции.

Пример. Написать программу для ввода-вывода информации о студенте.

class Cfio

{

char fio[20];

public:

void vvod()

{

cout<<"Enter the name"<<endl;

gets(fio);

}

void vivod()

{

cout<<"Name: "<<fio<<endl;

}

};

class Cngr

{

int ngr;

public:

void vvod()

{

cout<<"Enter group number"<<endl;

cin>>ngr;

}

void vivod()

{

cout<<"Group number: "<<ngr<<endl;

}

};

class Cstud

{

int nt;

Cfio fm;

Cngr gr;

public:

void vvod()

{

fm.vvod();

gr.vvod();

cout<<"Enter your phone number"<<endl;

cin>>nt;

}

void vivod()

{

fm.vivod();

gr.vivod();

cout<<"Phone number: "<<nt<<endl;

}

};

int main()

{

Cstud stud;

stud.vvod();

stud.vivod();

return 0;

}