

OOP

Lecture 21

23/12/2021

Assigning a Child Class object to Parent Class Pointer :

It is allowed to assign a child class dynamic object to a parent class pointer by simply writing,

```
Parent_class_name *ptr = new Child_class_name;
```

Suppose,

```
class P{  
-----  
};  
  
class C : public P{  
-----  
};
```

Then, it is valid and useful(in some cases) to write, **P *ptr = new C;**
But, we should remember that ptr cannot call any member function of child class C as it is a pointer of Parent Class. When creating any object, the first function which is called is the constructor. So, what will happen in this case? Which constructor will be called? Also, when we call a function using ptr, which function is called?

Check out this program to find out,

```

class P{
public:
    P(){    cout << "Parent Constructor\n"; }
    void show(){
        cout << "Show function of parent class\n";
    }
};
class C:public P{
public:
    C(){    cout << "Child Constructor\n"; }
    void show(){
        cout << "Show function of child class\n";
    }
};
int main()
{
    P *ptr=new C;
    ptr->show();
    return 0;
}

```

E:\program.exe

```

Parent Constructor
Child Constructor
Show function of parent class
-----

```

We can see that it is calling both constructors. We can similarly pass any argument/parameter as well to any constructor.

It is calling showing function of the parent class, it cannot call any function of the child class, if we try to do so, it will generate an error,

```

class P{
public:
    P(){    cout << "Parent Cons\n";    }
    void show(){    cout << "Show function of parent class\n";    }
};
class C:public P{
public:
    C(){    cout << "Child Cons\n"; }
    void show(){    cout << "Show function of child class\n";    }
    void show1(){    cout << "Show1 function of child class\n";    }
};
int main(){
    P *ptr=new C;
    ptr->show();
    ptr->show1();
    return 0;
}

```

resources Compile Log Debug Find Results X Close

am.cpp
im.cpp

Message

In function 'int main()':

[Error] 'class P' has no member named 'show1'

So, why do we need this??

We will come to this question later, first we should understand how to use it. Before going further, let's understand,

Function Overriding :

Function overriding in inheritance is giving the priority to any function over another function from another class. Like, in this case, even though ptr points to a child object, it is actually of Parent type. So, it calls the member function of the Parent class. In order to override the Base function instead of accessing it, we need to use virtual functions in the Base class.

Virtual Functions :

A virtual function is a member function in the parent class that we expect to be redefined in child classes. Basically, a virtual function is used in the parent class in order to ensure that the function is overridden. **virtual** is a keyword which is written at the start of a function prototype to make virtual functions.

A parent class pointer which is pointing to a child class object can access the virtual functions of the parent class which are redefined in the child class. Means that, if a function is written in the parent class and virtual keyword is also written with it (virtual function), and the same function is redefined in the child class as well, (same name, parameters and return type) only then, the parent class pointer can access the child class member function.

Sample Program

```
class P{
public:
    P(){          cout << "Parent Cons\n";    }
    virtual void show(){    cout << "Show function of parent class\n";    }
};
class C:public P{
public:
    C(){          cout << "Child Cons\n";    }
    void show(){    cout << "Show function of child class\n";    }
};
int main(){
    P *ptr=new C;
    ptr->show();
    return 0;
}
```

E:\program.exe

Parent Cons
Child Cons
Show function of child class

Process exited after 0.2309 seconds with return value 0
Press any key to continue . . .

In this way we can access the child class functions with the parent class pointer. Similarly, we also make the destructor of parent class virtual to call the destructor of the child class when the object is to be deleted.

```
class P{
public:
    P(){          cout << "Parent Constructor\n";    }
    virtual void show(){    cout << "Show function of parent class\n";    }
    virtual ~P(){          cout << "Parent Destructor\n";    }
};
class C:public P{
public:
    C(){          cout << "Child Constructor\n";    }
    //Function Overriding
    void show(){    cout << "Show function of child class\n";    }
    ~C(){          cout << "Child Destructor\n";    }
};
int main(){
    P *ptr = new C;
    delete ptr;
    return 0;
}
```

E:\program.exe

Parent Constructor
Child Constructor
Child Destructor
Parent Destructor

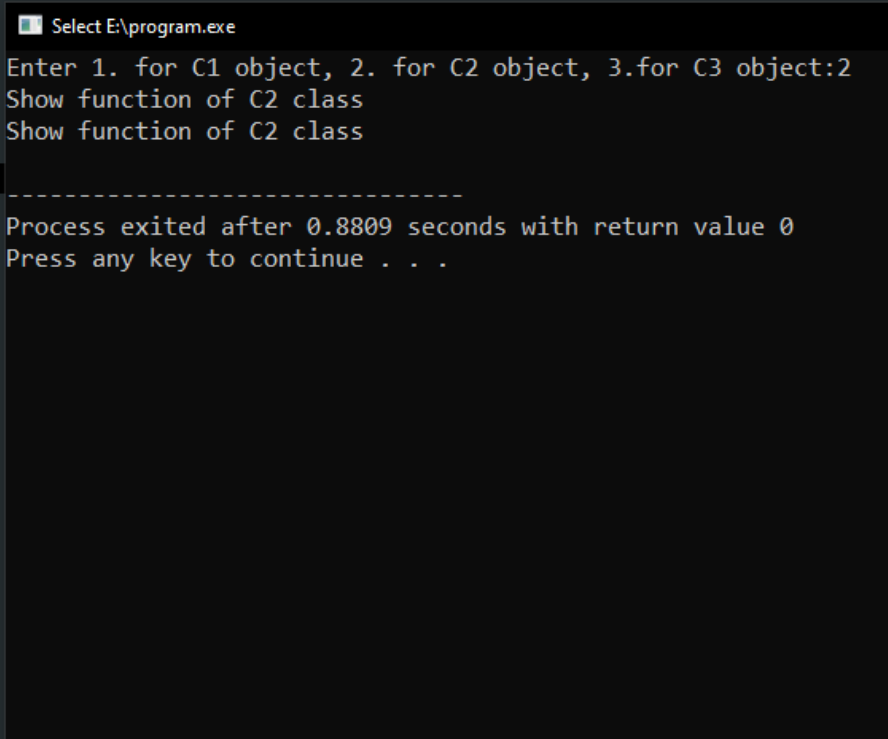
Process exited after 0.0 seconds with return value 0
Press any key to continue . . .

So, till now, there must be three conditions fulfilled in order to access the child class function with the parent class pointer, these are:

1. There must be Inheritance(Parent-Child relationship)
2. Virtual function in the Parent class
3. Overriding functions in the Child class.

We can use this technique in a situation where there are multiple childs of a parent class and we need to decide at runtime that object of which class is to be created. Consider an example when 3 child classes belong to a parent class, and the user decides which object is to be created. In normal working we will need to write a bunch of if-else if blocks to write the whole working in one block. Like this,

```
cout << "Enter 1. for C1 object, 2. for C2 object, 3. for C3 object:";
cin >> choice;
C1 *ptrC1;
C2 *ptrC2;
C3 *ptrC3;
if (choice == 1)
{
    ptrC1 = new C1;
    ptrC1->show();
    ptrC1->show();
    delete ptrC1;
}
else if (choice == 2)
{
    ptrC2 = new C2;
    ptrC2->show();
    ptrC2->show();
    delete ptrC2;
}
else
{
    ptrC3 = new C3;
    ptrC3->show();
    ptrC3->show();
    delete ptrC3;
}
return 0;
```

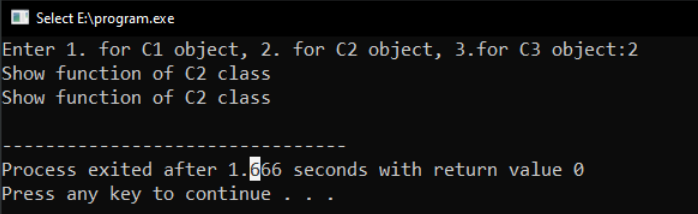


```
Select E:\program.exe
Enter 1. for C1 object, 2. for C2 object, 3. for C3 object:2
Show function of C2 class
Show function of C2 class
-----
Process exited after 0.8809 seconds with return value 0
Press any key to continue . . .
```

Here, we can clearly see that we need to write all of the work in each block. If there are 10 functions to be called then we need to write it again and again with objects of different classes in each if - else if block.

We can make this a lot easier by using the function overriding. We just need to make all the functions in the parent class virtual and then we are ready to go with the parent class pointer and the child class object of the user choice. Check out this code and compare it with the previous one,

```
int main()
{
    int choice;
    cout << "Enter 1. for C1 object, 2. for C2 object, 3. for C3 object:";
    cin >> choice;
    P *ptr;
    if (choice==1) ptr = new C1;
    else if (choice==2) ptr = new C2;
    else ptr = new C3;
    ptr->show();
    ptr->show();
    delete ptr;
    return 0;
}
```



Consider a real life example of a shooting game where there is an option of selecting a gun. There may be AK47, Glock 23, M4. All of these have different attributes yet the common thing is all of these are guns. So, consider a parent class is a Gun with common functions like reload(), fire() etc and all of the guns are its child classes. So, at runtime when the user selects a specific gun, only an object of that particular class is created and rest of the working is done only on the object of that class. The functions like fireSpeed(), damage() etc are made virtual in the parent class so, these functions would be accessed from the selected gun class.

-THE END-