OOP
Lecture 22

11/1/2021

Polymorphism :

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of something to have different shapes, in programming, having different functionalities. In C++, using a base class pointer or reference can call functions from object class.

In the previous lecture, we studied the concept of virtual functions and how to use them. In that case, we made the optional virtual functions in the parent class, such that those functions exist in parent class as virtual functions and the child classes may or may not override them, in order to change their functionality according to child class. It's up to child classes whether they want to override the virtual functions or not. If the child classes override them, the child class functions would be accessed otherwise the parent class functions would be called.

Force Child/Derived classes to override virtual functions :

We can force the child classes to override the virtual functions defined by the parent class by using Pure Virtual / Abstract Functions.

Pure Virtual / Abstract Functions :

A function in the parent class declared as virtual and having no implementation is called Pure Virtual / Abstract Functions.

When we write functions like this,

```
    virtual void function(int x)
    {       }
```

Either the curly braces are empty, it is still called the body of the function and we have the implementation of this function. If we want to write it in pure virtual/abstract form, we need to write,

virtual void function(int x) = 0;

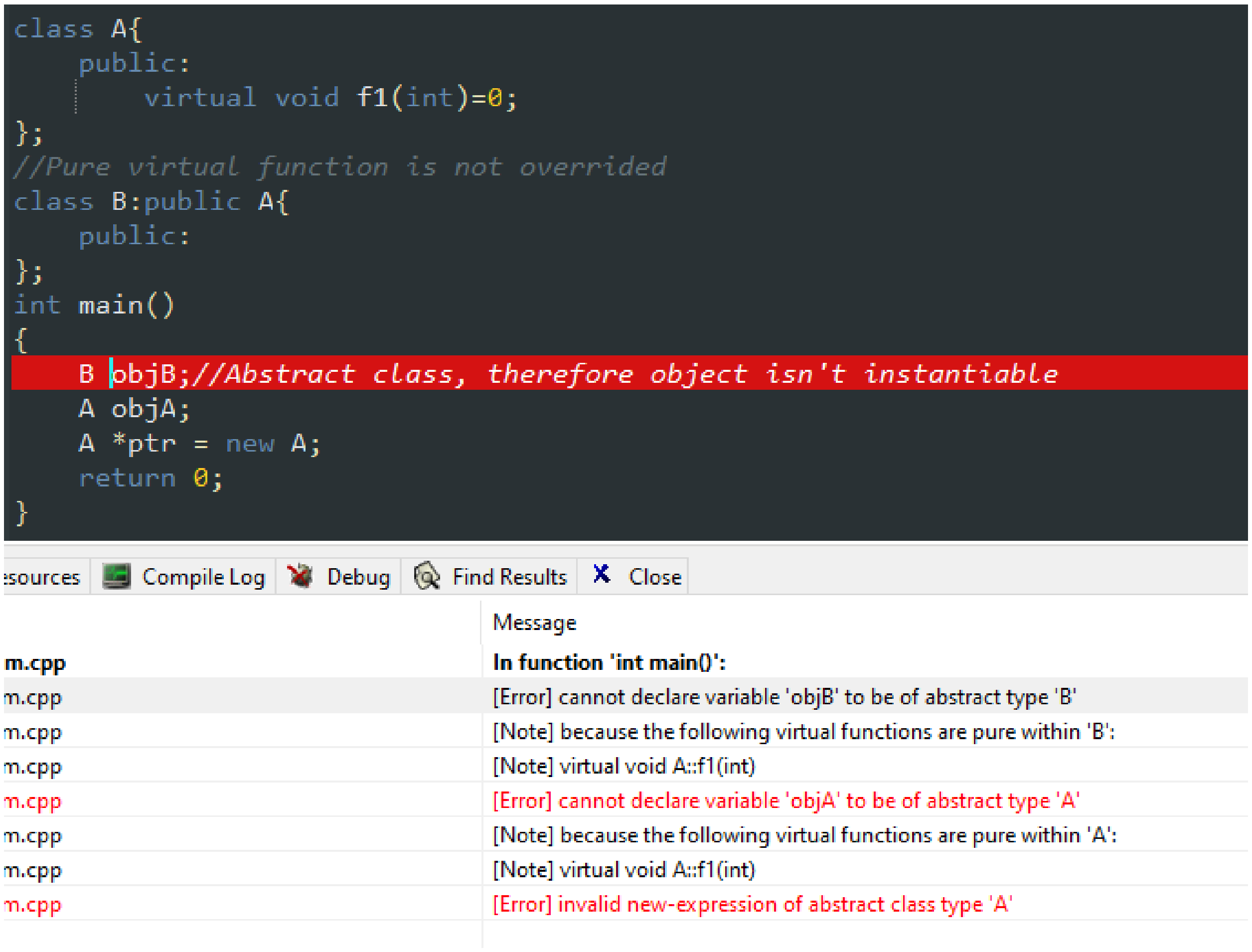
Now, this function does not have any implementation, so it is called as Pure Virtual / Abstract Function.

Abstract Class :
A class having one or more Pure Virtual / Abstract Function(s) is called an Abstract class.

Concrete Class :
A class with no pure/abstract virtual function is called a concrete class. It performs normal functionality of the class, like we were creating class till now.

Instantiation :
The process of creating an object of a class is called instantiation.

Instantiation is not possible in Abstract classes. It means that we cannot create any object of the class having any Pure Virtual / Abstract Function(s). As we know that the child inherits all the members(data members and functions) from its parent class. So, We ultimately need to create a child class and we must override that pure virtual function in the child class, otherwise the child class will also become an abstract class as it will also inherit the same Pure Virtual / Abstract Function(s) as its own members.



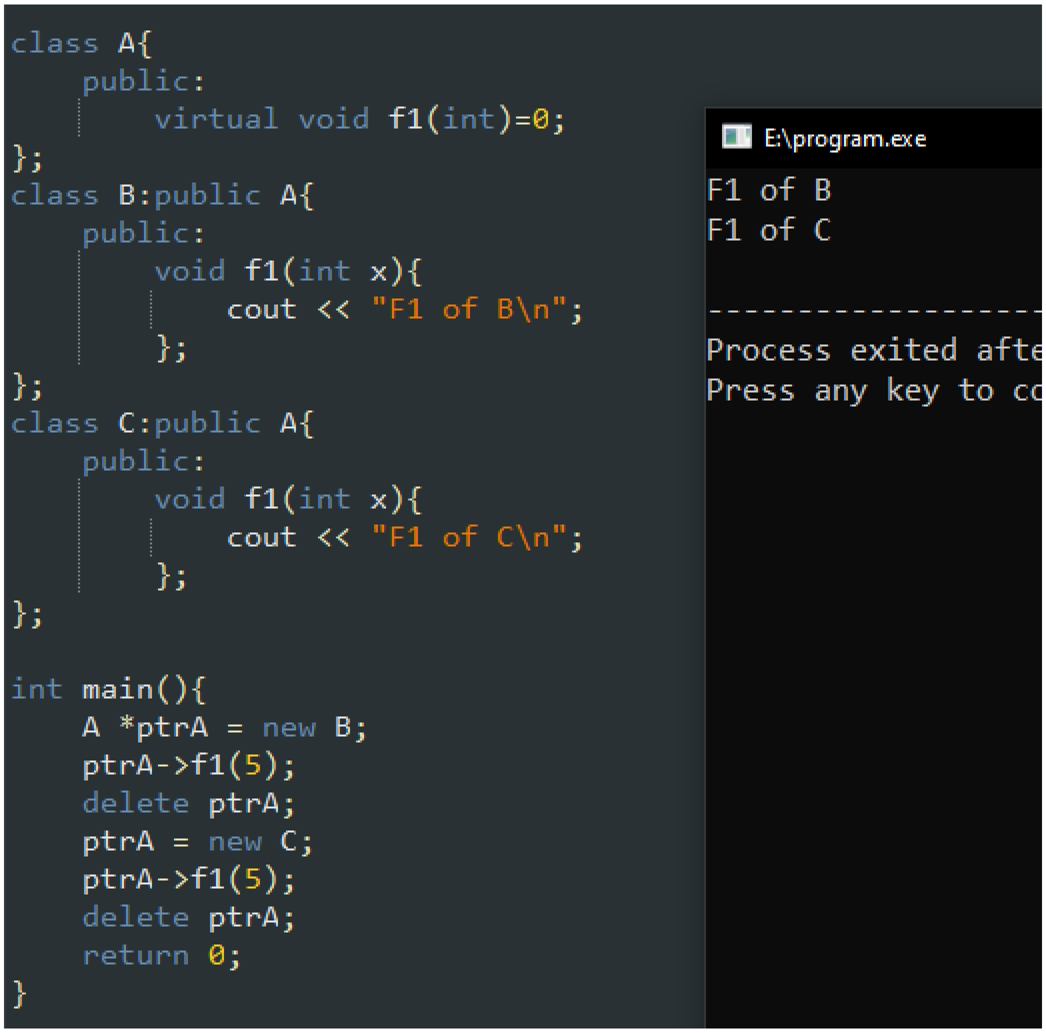
```
class A{
    public:
        virtual void f1(int)=0;
};
//Pure virtual function is not overrided
class B:public A{
    public:
};
int main()
{
    B objB;//Abstract class, therefore object isn't instantiable
    A objA;
    A *ptr = new A;
    return 0;
}
```

```
                                   Message
m.cpp                              In function 'int main()':
m.cpp                              [Error] cannot declare variable 'objB' to be of abstract type 'B'
m.cpp                              [Note] because the following virtual functions are pure within 'B':
m.cpp                              [Note] virtual void A::f1(int)
m.cpp                              [Error] cannot declare variable 'objA' to be of abstract type 'A'
m.cpp                              [Note] because the following virtual functions are pure within 'A':
m.cpp                              [Note] virtual void A::f1(int)
m.cpp                              [Error] invalid new-expression of abstract class type 'A'
```

The compiler is showing an error when we tried to create an object of the abstract class. All of these three lines are not correct as both of these classes are abstract until we override the function in the child class.

When we override this function in the child class we can call this function using the parent class pointer and base class object.

Check out this program,



```
class A{
    public:
        virtual void f1(int)=0;
};
class B:public A{
    public:
        void f1(int x){
            cout << "F1 of B\n";
        };
};
class C:public A{
    public:
        void f1(int x){
            cout << "F1 of C\n";
        };
};

int main(){
    A *ptrA = new B;
    ptrA->f1(5);
    delete ptrA;
    ptrA = new C;
    ptrA->f1(5);
    delete ptrA;
    return 0;
}
```

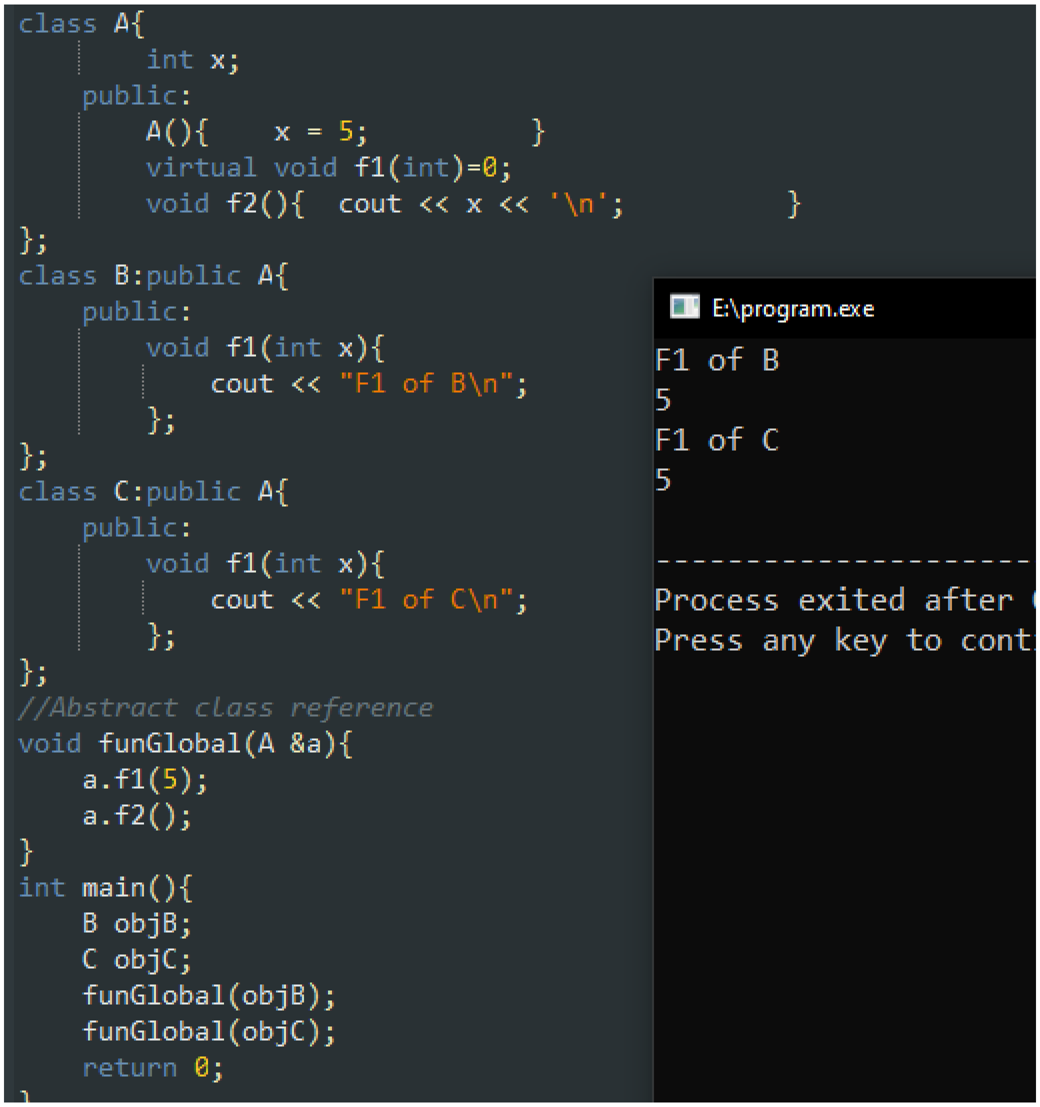
E:\program.exe

```
F1 of B
F1 of C

-------------------
Process exited afte
Press any key to co
```

Passing object to Global Functions :

When we use polymorphism in c++, there is another important feature associated with it. When we create abstract class pointers and child class objects (in a case when parent class has many child classes), we can make a generic function which accepts the parameters as **reference** of parent class type and we can pass any child class object to this function. In this function we can call any parent class function and the pure virtual / abstract functions of the parent class.



```cpp
class A{
        int x;
    public:
        A(){    x = 5;          }
        virtual void f1(int)=0;
        void f2(){  cout << x << '\n';          }
};
class B:public A{
    public:
        void f1(int x){
            cout << "F1 of B\n";
        };
};
class C:public A{
    public:
        void f1(int x){
            cout << "F1 of C\n";
        };
};
//Abstract class reference
void funGlobal(A &a){
    a.f1(5);
    a.f2();
}
int main(){
    B objB;
    C objC;
    funGlobal(objB);
    funGlobal(objC);
    return 0;
```

E:\program.exe

```
F1 of B
5
F1 of C
5


---------------------
Process exited after
Press any key to cont
```

What are the benefits of using polymorphism?
There are several benefits of using polymorphism. With this we can decide at runtime that the object of which class is to be created. And the further processing can be done using the pointer of parent class and object of the required class. It requires a single pointer/reference to handle multiple child classes. We can also make generic functions (1 function for many types) for all the child classes of a parent class. It will save us a lot of time and coding.

Back-end concepts how it is decided that which function is to call :
There is a concept called function binding. It controls the flow of the program. When we call a function, it is decided either at compile time or runtime that which function is to be called.

Compile time binding :

When a function is called, it changes the sequential flow of the program and jumps to the address of the function. Compile time binding are the decisions taken at the compilation of the program that which function is called. It happens when a single function exists or function overloading (compiler can differentiate functions with different parameters). So, it binds the function at the place from where it is called. It is also called Early binding or Static binding.

The problem arises when we have virtual functions. In this case, most probably, the object is defined and created at the run time. So, the compiler can not decide which function is to be called from that point because the object was not created till that point. So, for this scenario we have runtime binding.

Runtime Binding :

Runtime binding can be achieved by function overriding. There exists a V-Table in every class, which contains the list of functions that exist in every class. When we do function overriding, the V-Table binds the functions of the parent class with the child class. So, at runtime when the object is created and used to call a function, it is decided through V-Table, that which function is to be called based on the type of the object. It is also called Late binding or Dynamic binding.

-THE END-