girmiti software™

Introduction to C++

- Polymorphism
  - Compile time Polymorphism
  - Run time Polymorphism
- Virtual functions
- Pure virtual functions

Polymorphism

Polymorphism:

Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and morphs means forms. The word polymorphism means one having many forms. It refers to identically named methods that have different behavior depending on the type of object they refer.

Type of polymorphism:

- Static or Compile time polymorphism(Early binding)
- Dynamic or Run time polymorphism(Late binding)

Polymorphism

Compile time polymorphism:

The process of choosing the method at compile time is known as compile time polymorphism. During compile time,the C++ compiler determines which function is used based on the parameter passed to the function.The compiler then substitutes the correct function for each invocation.Such compiler based substitutions are called static linkage.

Polymorphism

Example of compile time polymorphism:

```
#include<iostream>
using namespace std;
class Super
{           public:
            void funBase() { cout << "Super function"<<endl; }
};
class Sub : public Super
{           public:
            void funBase() { cout << "Sub function"<<endl; }
};
int main()
{ Super* ptr;       // Super class pointer
 Sub obj;
 ptr = &obj;
 ptr->funBase();
}
```

Output:

```
Super function
```

Polymorphism

As we can see in the example,even though function call is made from the base class pointer to access members of the derived class,the function call has not reached to the derived class members.so the base class members have been displayed not the derived class.so C++ compiler takes by default only static binding.

Polymorphism

Run time polymorphism:

Runtime polymorphism is a process in which a call to a method is resolved at runtime rather than compile-time. Sometimes compiler can't know which function will be called till program is executed (runtime). Hence, now compiler determines the type of object at runtime, and then binds the function call.
In C++ Run time polymorphism can be achieved by using virtual function.

C++ allows binding to be delayed till run time. When we have a function with same name, equal number of arguments and same data type in same sequence in base class as well derived class and a function call of form:

```
base_class_type_ptr->member_function(args);
```

will always call base class member function. The keyword virtual on a member function in base class indicates to the compiler to delay the binding till run time.

Virtual functions

Virtual function:

- Virtual Function is a function in base class, which is overridden in the derived class, and which tells the compiler to perform Late(run time) Binding on this function.
- virtual keyword is used to make a member function of the base class virtual.
- When we want to use same function name in both the base and derived class, then the function in base class is declared as virtual by using the virtual keyword and again re-defined this function in derived class without using virtual keyword.
- When derived class overrides the base class method by redefining the same function, then if we wants to access redefined the method from derived class through a pointer from base class object, then you must define this function in base class as virtual function.

Syntax:

```
virtual return_type function_name()
{
..... .....
}
```

Virtual function

Example using virtual keyword:

We can make base class's methods virtual by using virtual keyword while declaring them. virtual keyword will lead to Late Binding of that method:

```
#include<iostream>
using namespace std;
class A
{
public:
virtual void show()
{
cout<<"Hello base class"<<endl;
} };
class B : public A
{ public:
void show()
{
cout<<"Hello derive class"<<endl;
}
};
```

Virtual function

```
int main()
{
A aobj;
B bobj;
A *bptr;
bptr=&aobj;
bptr->show();    // call base class function
bptr=&bobj;
bptr->show();    // call derive class function
}
```

Output:

```
Hello base class
Hello derive class
```

Virtual function

Accessing Private Method of Derived class using virtual keyword:

We can call private function of derived class from the base class pointer with the help of virtual keyword. Compiler checks for access specifier only at compile time. So at run time when late binding occurs it does not check whether we are calling the private function or public function.

Example:

```
#include<iostream>
using namespace std;
class A
{
 public:
virtual void show()
{
cout << "Base class\n";
}
};
class B: public A
{
private:
```

Virtual function

```
virtual void show()
{
cout << "Derived class\n";
}
};
int main()
{
A *a;
B b;
a = &b;
a ->show();
}
```

Output:

Derived class

Pure virtual function

Pure virtual function:

A pure virtual function is a type of function which has only a function declaration.It does not have the function definition.

Sometimes we want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition we could give for the function in the base class,we can make this as pure virtual function.

Pure virtual function

A pure virtual function can also have the following format,when a virtual function is declared within the class declaration itself,the virtual function may be equated to zero if it does not have a function definition . They start with virtual keyword and ends with= 0 as shown below:

```
class base
{
…
virtual void getdata()=0;  //pure virtual function
…
}
```

Pure virtual function

Example:

```
#include<iostream>
using namespace std;
class Bank
{
        public:
        virtual int getInterestRate()=0;
};
class Canara:public Bank
{
        private:
        int getInterestRate(){return 5;}
};
class Axis:public Bank
{
        public:
        int getInterestRate(){return 7;}
};
```

Pure virtual function

```
int main()
{
        Bank *b;
        Canara c;
        Axis a;
        b=&a;
        cout<<"Axis Bank rate of Interest:"<<b->getInterestRate()<<endl;
        b=&c;
        cout<<"Canara bank rate of Interest:"<<b->getInterestRate()<<endl;
}
```

Output:

Axis Bank rate of Interest:7

Canara bank rate of Interest:5

Discussions