Pure Virtual Functions and Abstract Classes in C++

Difficulty Level : Medium • Last Updated : 27 Mar, 2021

Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class. For example, let Shape be a base class. We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly an Animal class doesn't have implementation of move() (assuming that all animals move), but all animals must know how to move. We cannot create objects of abstract classes.
A pure virtual function (or abstract function) in C++ is a virtual function for which we can have implementation, But we must override that function in the derived class, otherwise the derived class will also become abstract class (For more info about where we provide implementation for such functions refer to this https://stackoverflow.com/questions/2089083/pure-virtual-function-with-implementation). A pure virtual function is declared by assigning 0 in declaration. See the following example.

CPP

```
// An abstract class
class Test
{
    // Data members of class
ublic:
    // Pure Virtual Function
    virtual void show() = 0;
```

```
   /* Other members */
};
```

A complete example:

A pure virtual function is implemented by classes which are derived from a Abstract class. Following is a simple example to demonstrate the same.

CPP

```
#include<iostream>
using namespace std;

class Base
{
   int x;
public:
    virtual void fun() = 0;
    int getX() { return x; }
};

// This class inherits from Base and implements fun()
class Derived: public Base
{
    int y;
public:
    void fun() { cout << "fun() called"; }
};

int main(void)
{
    Derived d;
    d.fun();
    return 0;
}
```

Output:

```
fun() called
```

me Interesting Facts:

) *A class is abstract if it has at least one pure virtual function.*

In the following example, Test is an abstract class because it has a pure virtual function show().

C

```
// pure virtual functions make a class abstract
#include<iostream>
using namespace std;

class Test
{
   int x;
public:
    virtual void show() = 0;
    int getX() { return x; }
};

int main(void)
{
    Test t;
    return 0;
}
```

Output:

```
Compiler Error: cannot declare variable 't' to be of abstract
 type 'Test' because the following virtual functions are pure
within 'Test': note:     virtual void Test::show()
```

2) *We can have pointers and references of abstract class type.*
For example the following program works fine.

CPP

```
#include<iostream>
using namespace std;

class Base
{
```

```
public:
    virtual void show() = 0;
};

class Derived: public Base
{
public:
    void show() { cout << "In Derived \n"; }
};

int main(void)
{
    Base *bp = new Derived();
    bp->show();
    return 0;
}
```

Output:

```
In Derived
```

3) *If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.*
The following example demonstrates the same.

CPP

```
#include<iostream>
using namespace std;
class Base
{
public:
    virtual void show() = 0;
};

class Derived : public Base { };

int main(void)
{
  Derived d;
  return 0;
```

```
Compiler Error: cannot declare variable 'd' to be of abstract
'Derived'  because the following virtual functions are pure wi
'Derived': virtual void Base::show()
```

4) *An abstract class can have constructors.*

For example, the following program compiles and runs fine.

CPP

```
#include<iostream>
using namespace std;

// An abstract class with constructor
class Base
{
protected:
int x;
public:
virtual void fun() = 0;
Base(int i) {
              x = i;
            cout<<"Constructor of base called\n";
            }
};

class Derived: public Base
{
    int y;
public:
    Derived(int i, int j):Base(i) { y = j; }
    void fun() { cout << "x = " << x << ", y = " << y<<'\n'; }
};

int main(void)
{
    Derived d(4, 5);
    d.fun();

  //object creation using pointer of base class
    Base *ptr=new Derived(6,7);
      ptr->fun();
    return 0;
```

Output:

```
Constructor of base called
x = 4, y = 5
Constructor of base called
x = 6, y = 7
```

Comparison with Java
In Java, a class can be made abstract by using abstract keyword. Similarly a function can be made pure virtual or abstract by using abstract keyword. See Abstract Classes in Java for more details.
Interface vs Abstract Classes:
An interface does not have implementation of any of its methods, it can be considered as a collection of method declarations. In C++, an interface can be simulated by making all methods as pure virtual. In Java, there is a separate keyword for interface.

We can think of Interface as header files in C++, like in header files we only provide the body of the class that is going to implement it. Similarly in java in Interface we only provides the body of the class and we write the actual code in whatever class implements it.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.