Lecture 21

Abstract Classes in C++

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.

They are classes that can only be used as base classes, and thus are allowed to have virtual member functions without definition (known as pure virtual functions). The syntax is to replace their definition by =0 (an equal sign and a zero):

An abstract base Polygon class could look like this:

```
// abstract class CPolygon
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area () =0;
};
```

Notice that area has no definition; this has been replaced by =0, which makes it a *pure virtual function*. Classes that contain at least one *pure virtual function* are known as *abstract base classes*.

Some interesting facts about abstract class

1) We can't create an object of abstract class.

```
// 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.

```
#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.

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

class Derived : public Base {
};

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

output:

Compiler Error: cannot declare variable 'd' to be of abstract type

'Derived' because the following virtual functions are pure within

'Derived': virtual void Base::show()

Example:

```
#include <iostream>
using namespace std;
class Test {
protected:
   int width, height;

public:
   void set_values(int a, int b)
   {
      width = a;
      height = b;
   }
   virtual int area(void) = 0;
```

```
};
class r : public Test {
public:
   int area(void)
   {
      return (width * height);

   }
};
class t : public Test {
public:
   int area(void)
   {
      return (width * height / 2);
   }
};
int main()
{
   r rect;
   t trgl;
   Test* ppoly1 = &rect;
   Test* ppoly2 = &trgl;
   ppoly1->set_values(4, 5);
   ppoly2->set_values(4, 5);
   cout << ppoly1->area()<<endl;
   cout << ppoly2->area();
   return 0;
}
```

output:

20

10

Explanation: In this program, we are calculating the area of rectangle and triangle by using abstract class.