**Polymorphism**

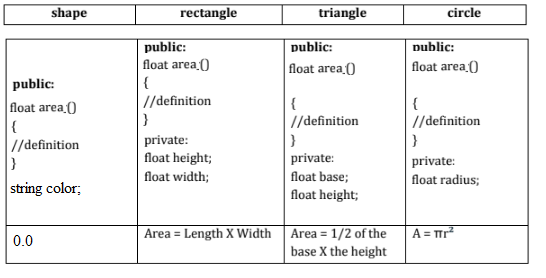
**Objectives:**

* Use base class pointers to call derived class objects
* Declare virtual functions and destructors
* See polymorphism in action
* Pure virtual function.
* Dynamic down casting

*NOTE: USE THE GIVEN CODE AS REFERENCE. DO NOT JUST COPY PASTE AS THERE CAN BE SYNTAX ERRORS IN GIVEN CODE SNIPPETS.*

**Exercise 0 Creating Inheritance relationship:**

* Create a class shape with functions and member as shown in figure
* Create classes circle square and rectangle as sub classes of class shape
* Each sub class of shape should override a calculate area of class shape according the given formula in figure.
* Your classes should have overloaded constructors that will take the member variables as input.



**Exercise 1 Test Case:**

In your main function, create following objects

triangle t1(1.0,9.0, “Red”)

circle c1(2, “Blue”);

rectangle r1(6,2, “Orange”)

cout<<t1.area();

cout<<t1.color;

cout<<c1.area();

cout<<r1.area();

shape \*sptr1= &t1;

shape &sref=r1;

cout<< sptr->area()

cout<< sptr->color;

cout<< sref.color;

cout<< sref.area();

You can see that the call to area using sptr or sref invoked shape::area function

**Exercise 2 Virtual functions:**

What if we wanted to use the definition of the derived class function?

To accomplish this, we can add the keyword virtual to the declaration of the area() method in the shape class.

Specifying a function as virtual makes sure that whenever we use a base class pointer/ref pointing to an object of a derived class to call a function, the definition of the method declared in the derived class is used.

Modify the area method of the shape class as shown below, compile your code, execute it and paste the output in the space provided below

virtual float area()

Output:

**Exercise 3a Use of Polymorphism, parent pointer as function parameter:**

In the above exercises, we have seen a very simple implementation of Polymorphism. The real power of this feature is realized when we have a collection of objects of multiple derived classes and we use a pointer of the base class to call their respective overloaded methods. Or you can pass derived class object to a function with formal parameter of base class type.

Create a function sumArea that takes two shapes of any type and return the sum of their area (shape/rectangle/circle etc...). Note that sumArea is a nonmember function, you can declare is above your main function. Prototype of function is given.

int sumArea(? Shape1,? Shape2)

{

//

}

//test sumArea in main

int main

{

triangle t1(1.0,9.0, “Red”)

circle c1(2, “Blue”);

rectangle r1(6,2, “Orange”)

shape s1(“Purple”);

cout<<sumArea(t1,c1);

cout<<sumArea(s1,r1);

cout<<sumArea(s1,t1);

}

**Exercise 3b Use of polymorphism, keeping sibling types in same array:**

Another advantage of polymorphism is to keep the object of different sub types of same parent class in one array. The array will be on parent class pointer type.

As an exercise create take 5 shapes as input from user. Store these in one array.

In one loop print the area of these shapes.

Use following pseudo code to take input from user (although a better way will be to overload extraction operator)

int count= 5

? shapesArray=?

for(int i=0; i<count)

{

cout << "Press 1 for a triangle, 2 for rectangle and 3 for a circle." << end;

switch (getch())

{

case '1':

//get base from user as input

//get height from user as input

//get color from user as input

// create new triangle object and add to shapesArray[i]

i++

break;

case '2':

//get length from user as input

//get width from user as input

//get color from user as input

// create new rectangle object and add to shapesArray[i]

i++

break;

case '3':

//get radius from user as input

// create new circle object and add to shapesArray[i]

i++

break;

default:

cout<<"Invalid input. Enter again." <<endl<<endl;

break;

}

//print area of all the shapes in shapeArray

//delete all object you have created using new.

**Exercise 4 Virtual Destructors:**

Although things seem to be fine on the surface, there is a problem in the program we just wrote. To observe this problem, we must add destructors for all classes. Add the following inline definitions of the destructors in their corresponding classes

~shape(){ cout << "~shape() called."<<endl; }

~triangle(){ cout << "~ triangle () called."<<endl; }

~circle() { cout << "~ circle () called."<<endl; }

~rectangle(){ cout << "~rectangle() called."<<endl; }

Execute the following lines in main and see output below.

shape \*s1= new triangle(1.0,9.0, “Red”);// constructor of triangle invoked

delete s1; //identify which destructor in invoked

Can you see what went wrong? When using delete to deallocate memory, only the base class destructor is called whereas the derived class destructor is not called at all. This can cause a memory leak issue. To avoid this we declare the base class destructor as virtual. Doing this will make sure that the derived class destructor is called even if you are using a base class pointer to call the destructor. Now change the definition of the base class destructor to make it virtual, execute the program and paste the output in the box given below. Make sure you can see the derived class destructors being called in the output.

AS A RULE OF THUMB, ANY CLASS WITH VIRTUAL FUNCTION SHOULD HAVE VIRTUAL DESTRUCTOR.

**Exercise 5 Pure Virtual Function:**

Write the following line in main and see if you can create and object of type shape

Shape S1(“red”);

Now make shape.area() a pure virtual function, run your code again to see if you can create a shape type object now.

**Exercise 6 Dynamic casting:**

Create a class Sphere that inherits from Circle.

Add a new function volume () in class sphere.

The volume function should return volume of sphere calculate using formula

**V=4/3πr3**

Note: sphere inherits r from circle.

Create a function checkShape in driver file that will take any shape, it will print its color and area, if the shape is sphere it will also print its volume.

Helper code given below

void checkShape(Shape \*S)

{

// print color and area of shape

// dynamic cast S to sphere type pointer to see if shape is sphere by checking if it returns null ptr

// if no print volume of sphere

}

**//check the function using following function in main**

triangle t1(1.0,9.0, “Red”)

circle c1(2, “Blue”);

rectangle r1(6,2, “Orange”)

sphere s1(4, “green”)

checkShape(&t1);

checkShape(&c1);

checkShape(&r1);

checkShape(&s1);