CPS235 Object Oriented Programming in C++

LAB 9 BESE-15A *16th May 2010*

LAB9 Polymorphism

Objectives

By the end of the lab, you should be able to

- Understand the concept of polymorphism
- Declare and use virtual and pure virtual functions to effect polymorphism
- Distinguish between abstract and concrete classes.
- Use virtual destructors to ensure that all appropriate destructors run on an object
- Use run-time type information (RTTI) with downcasting, dynamic_cast, typeid and type_info

NOTES

New Operator:

Remember that new operator allocates memory on Heap and returns the pointer. Hence the value returned by new operator would be a pointer value. So doing following is an error:

```
Circle X = new Circle () ; // X is just an object not a pointer.
```

You should do one of the following:

```
Circle *px = new Circle ();
```

or

```
Circle x ();
```

Of course, with px, you should access the variable using arrow operator. Something like px->x, px->y, and px->r. And, this is assuming that there exists a default constructor for Circle. If you have constructor Circle(int, int , int) then, you should create object like this:

```
Circle x ( 1, 2, 3);
```

```
Circle *px = new Circle ( 1, 2, 3 );
```

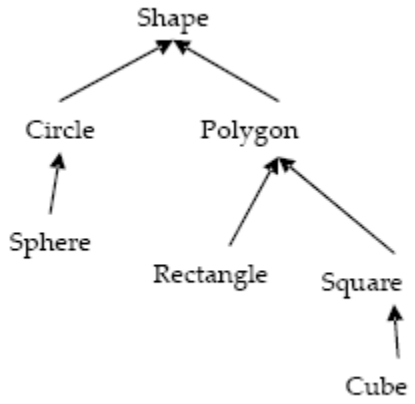
Base Class Constructor:

You should be familiar with calling base class constructors from derived class constructor. Here is an example.

```
class Figure {

public:
    Figure (int xx, int yy) {
        x = xx;
        y = yy;
    }

protected:
    int x;
    int y;
};

class Rectangle : public Figure {
public:

    /* Call Constructor of Figure first and then construct this
     * object
     */
    Rectangle (int xx, int yy, int ll, int hh) : Figure(xx, yy) {
        length = ll;
        height = hh;
    }

protected:
    int length;
    int height;
};
```

Question 1

In this question, you have to implement the following class hierarchy using Inheritance.



You have to define one class for each shape. And, use *public* inheritance when deriving from base classes. Each shape should contain following attributes and member functions.

```
Circle
    int x, y, r;
    void area();
    void perimeter();
    void volume();
Sphere
    int x, y, r;
    void area();
    void perimeter();
    void volume();
Rectangle
    int length, width;
    void area();
    void perimeter();
    void volume();
Square
    int length;
    void area();
    void perimeter();
    void volume();
Cube
    int length;
    void area();
    void perimeter();
    void volume();
```

Functions area(), perimeter(), and volume() are inherited from Shape class. You have to declare them as virtual functions. Since classes Shape and Polygon are conceptual, you have to define them as abstract classes (remember pure virtual functions). All others should be concrete classes i.e., all classes should inherit these functions from its base class and override.

Certain functions are not defined for certain shapes. For example, volume is not defined for Circle and, hence, print a warning message in the overridden function. Something like following:

```
int Circle::volume ( void ) {
    cout << "Volume is not defined for circle." << endl;
}
```

Along with these functions, you should have proper constructors to construct the objects of each class. Please note that, you can only create objects of concrete classes. You should use appropriate formulae to calculate the area, perimeter and volume. You can take the value of PI to be 3.14. All methods should print appropriate messages. For example, “Area of circle is 31.0” etc.

Sample Test Program

```
main ( ) {

Shape *ptr;
Circle c ( 1, 2, 5 );
Rectangle r ( 5, 10 );

    ptr = &c;
    ptr->perimeter();

    ptr = &r;
    ptr->area();
    ptr->volume();

} // main
```

Sample Output:

```
Perimeter of circle = 31.4
Area of rectangle = 50
Volume is not defined for rectangle.
```

Question 2:

Use the `Package` inheritance hierarchy created in homework assignment 1 to create a program that displays the address information and calculates the shipping costs for several `Packages`. The program should contain an `array` of `Package` pointers to objects of classes `TwoDayPackage` and `OvernightPackage`. Loop through the `array` to process the `Packages` polymorphically. For each `Package`, invoke get functions to obtain the address information of the sender and the recipient, then print the two addresses as they would appear on mailing labels. Also, call each `Package`'s `calculateCost` member function and print the result. Keep track of the total shipping cost for all `Packages` in the `array`, and display this total when the loop terminates.

Question 3:

Develop a polymorphic banking program using the `Account` hierarchy created in Lab4. Create an `array` of `Account` pointers to `SavingsAccount` and `CheckingAccount` objects. For each `Account` in the `array`, allow the user to specify an amount of money to withdraw from the `Account` using member function `debit` and an amount of money to deposit into the `Account` using member function `credit`. As you process each `Account`, determine its type. If an `Account` is a `SavingsAccount`, calculate the amount of interest owed to the `Account` using member function `calculateInterest`, then add the interest to the account balance using member function `credit`. After processing an `Account`, print the updated account balance obtained by invoking base class member function `getBalance`.

For this program, you will need to use the typeid operator to determine the type of the elements of the account array at runtime. The following code demonstrates the use of the typeid operator. You will have to include the header file <typeinfo> to make this work.
Note: For this example to run in Visual C++ , you need to enable RTTI (Run-Time Type Info) for the project

```
class Base
   {
   virtual void virtFunc()      //needed for typeid
      {  }
   };
class Derv1 : public Base
   { };
class Derv2 : public Base
   { };
////////////////////////////////////////////////////////////////
void displayName(Base* pB)
   {
   cout << "pointer to an object of ";  //display name of class
   cout << typeid(*pB).name() << endl;  //pointed to by pB
   }
//-----------------------------------------------------------
int main()
   {
   Base* pBase = new Derv1;
   displayName(pBase);   //"pointer to an object of class Derv1"

   pBase = new Derv2;
   displayName(pBase);   //"pointer to an object of class Derv2"
   return 0;
   }
```