TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PURWANCHAL CAMPUS

DHARAN

LABSHEET 7 : [Virtual Function, Virtual Base Class, and RTTI]

BY

Rabin Poudel [PUR079BCT056]

DEPARTMENT OF COMPUTER & ELECTRONICS ENGINEERING

PURWANCHAL CAMPUS

DHARAN, NEPAL

JULY, 2024

0.1 Question 1 :

Write a program to create a class shape with functions to find area of the shapes and display the name of the shape and other essential component of the class. Create derived classes circle, rectangle and trapezoid each having overridden functions area and display. Write a suitable program to illustrate virtual functions and virtual destructor.

Code:

```
#include <iostream>
#include <cstring>
#define pi 3.1415
using namespace std;
class shape
{
 protected:
   string sname;
   float sarea;
 public:
   shape()
   {
      sname = "shape";
      sarea = 0;
   }
   shape(float a, string n="shape")
   {
      sname = n;
      sarea = a;
   }
   virtual float area()
   {
      return sarea;
   }
   string name()
   {
      cout << "Shape " << sname << endl;
      return sname;
   }
   virtual ~shape()
```

```
    {
       cout << "Destructor of Shape " << endl;
    }
};
class circle:public shape
{
  protected:
    float radius;
  public:
    circle(int r, string n = "circle")
    {
       radius = r;
       sname = n;
    }
    float area()
    {
       sarea = pi    *  radius   *  radius;
       return shape::area();
    }
    string name()
    {
       cout << "Circle " << sname << endl;
       return sname;
    }
     ~circle()
    {
       cout << "Circle destructor" << endl;
    }

};
class rectangle:public shape
{
  protected:
    float length, breadth;
  public:
    rectangle(float l, float b, string
         n="rectangle"):length(l),breadth(b)
    {
       sname = n;
```

```
    }
    float area()
    {
       sarea = length   *breadth;
       return shape::area();
    }
    string name()
    {
       cout << "Rectangle " << sname << endl;
       return sname;
    }
     ~rectangle()
    {
       cout << "Rectangle destructor" << endl;
    }
};
class trapezoid:public shape
{
 protected:
   float paralleside[2];
   float nonparallelside[2];
 public:
   trapezoid(float a1, float a2, float b1, float b2, string n=
         "Trapezoid")
   {
       paralleside[0] = a1;
       paralleside[1] = a2;
       nonparallelside[0] = b1;
       nonparallelside[1] = b2;
       sname = n;
   }
   float area()
   {
       sarea =
            (paralleside[0]+paralleside[1])/2.0          *(nonparallelside[0]+nonparallelside[1])/2.0;
       return shape::area();
   }
   string name()
   {
```

```
        cout << "Trapezoid " << sname << endl;
        return sname;
    }
     ~trapezoid()
    {
        cout << "Trapezoid destructor" << endl;
    }
};
int main()
{
    shape  *sh;
    sh = new circle(4,"ball");
    sh->name();
    cout << sh->area() << endl;
    delete(sh);
    sh = new trapezoid(200,400, 100 , 100,"fancy stadium");;
    sh->name();
    cout << sh->area() << endl;
    delete(sh);
    sh = new rectangle(240,240,"ground");;
    sh->name();
    cout << sh->area() << endl;
    delete(sh);
    return 0;
}
```

0.2 Question 2:

Create a class Person and two derived classes Employee, and Student, inherited from class Person. Now create a class Manager which is derived from two base classes Employee and Student. Show the use of the virtual base class.

Code:

```
#include <iostream>
#include <cstring>
using namespace std;
class person
{
 private:
   string name;
 public:
   person(){}
   person(string n)
   {
      name = n;
   }
   string getname()
   {
      return name;
   }
};
class employee: virtual public person
{
 private:
   int salary;
 public:
   employee(string n, int s):person(n),salary(s){};
   int getsalary()
   {
      return salary;
   }
};
class student: virtual public person
{
```

```
  private:
    string major;
  public:
    student(string n, string m):person(n),major(m){};
    string getmajor()
    {
       return major;
    }
};
class manager:public employee, public student
{
  private:
  public:
    manager(string n, string m, int s): employee(n,s), student(n,m),
          person(n)
    {
    };
};
int main()
{
    manager demo("Acd Efgh","BCT",50000);
    cout << "Name: " << demo.getname() << endl;
    cout << "Major: " << demo.getmajor() << endl;
    cout << "Salary: " << demo.getsalary() << endl;
    return 0;
}
```

0.3 Question 3:

Write a program with Student as abstract class and create derive classes Engineering, Medicine and Science from base class Student. Create the objects of the derived classes and process them and access them using array of pointer of type base class Student.

Code:

```
#include <iostream>
#include <cstring>
using namespace std;
class student
{
 private:
 protected:
   string name;
   int rank;
 public:
   virtual string getname() = 0;
   virtual int getrank() = 0;
};
class engineering : public student
{
 private:
 public:
   engineering(string n,int r){
      name= n;
      rank=r;
   }
   string getname()
   {
      return name;
   }
   int getrank()
   {
      return rank;
   }
};
```

```
class medicine: public student
{
 private:
 public:
   medicine(string n,int r){
      name=n;
      rank=r;
   }
   string getname()
   {
      return name;
   }
   int getrank()
   {
      return rank;
   }
};
class science : public student
{
 private:
 public:
   science(string n, int r){
      name = n;
      rank = r;
   }
   string getname()
   {
      return name;
   }
   int getrank()
   {
      return rank;
   }
};
int main()
{
   student *  st[3];
   st[0] = new engineering("Abcd", 3);
   st[1] = new medicine("Efgh",1);
```

```
    st[2] = new science("ljkl",2);
    cout << "Student of various field" << endl;
    for (int i = 0; i < 3; ++i)
    {
       cout << "Name " << st[i]->getname() << endl;
       cout << "Rank " << st[i]->getrank() << endl;
    }
    return 0;
}
```

0.4 Question 4:

Create a polymorphic class Vehicle and create other derived classes Bus, Car and Bike from Vehicle. With this program illustrate RTTI by the use of dynamic cast and typeid operators.

Code:

```
#include <iostream>
#include <cstring>
#include <typeinfo>
using namespace std;
class vehicle
{
 private:
 protected:
   string registration;
   int noofwheels;
 public:
   vehicle(string r, int n)
   {
      registration = r;
      noofwheels = n;
   }
   string getregistration()
   {
      cout << "Vehicle getRegistratin called" << endl;
      return registration;
   }
};
class bus : public vehicle
{
 private:
 public:
   bus(string r):vehicle(r,4){};
   string getregistration()
   {
      cout << "Bus getRegistratin called" << endl;
      return registration;
```

```
    }
};
class car : public vehicle
{
 private:
 public:
   car(string r):vehicle(r,4){};
   string getregistration()
   {
      cout << "Car getRegistratin called" << endl;
      return registration;
   }
};
class bike : public vehicle
{
 private:
 public:
   bike(string r):vehicle(r,2){};
   string getregistration()
   {
      cout << "Bike getRegistratin called" << endl;
      return registration;
   }
};

int main()
{
   vehicle    *vlist[3];
   bus  *bs = new bus("1");
   car   *c = new car("1");
   bike   *b = new bike("1");
   vlist[0] = dynamic_cast<vehicle         *>(bs);
   vlist[1] = dynamic_cast<vehicle         *>(c);
   vlist[2] = dynamic_cast<vehicle         *>(b);
   for(int i = 0; i < 3 ; i++)
   {
      cout << typeid(    *vlist[i]).name() << endl;
      cout << vlist[i]->getregistration() << endl;
   }
```

```
    cout << typeid(   *bs).name() << endl;
    cout << typeid(   *c).name() << endl;
    cout << typeid(   *b).name() << endl;
    return 0;
}
```