

Lab Sheet 7

Understanding the Concept of Virtual function, Virtual base class and

RTTI

Virtual Function

The overridden function in the derived class can be invoked by means of a base class pointer if the function is declared virtual in the base class. Suppose a virtual function `get()` is defined in the base class `Base` and again it is defined in the derived class `Derived`.

We can use the base class pointer to invoke the `get()` function of the derived class.

```
Derived d;
```

```
Base *b;
```

```
b=&d;
```

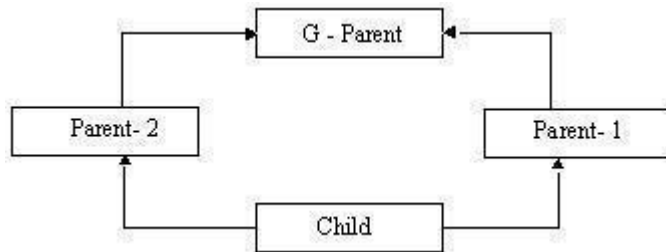
```
b-> get( ) //it calls the get ( ) function of the  
derived
```

```
class.
```

Virtual Destructors

When a base class pointer that is pointing to a derived class object is deleted, destructor of the derived class as well as destructors of all its base classes is invoked, if the destructor in the base class is declared as virtual.

Virtual Base Class



In this type of inheritance there may be ambiguity in the members of the derived class child because it is derived from two base classes, which are again derived from the same base class. Hence to avoid this ambiguity the class G - parent can be made virtual.

Runtime Type Information (RTTI)

The runtime type information is one of the features of C++ that exhibit runtime polymorphic behavior. In C++ we can find the type information of an object at runtime and change the type of the object at runtime. The operators `dynamic_cast` and `typeid` are used for runtime type information.

For example if `Animal` is a polymorphic base class and `Dog` and `Cat` are derived classes of base class `Animal` then

```
Animal *anmp;
```

```
Dog dg;
```

```
Cat ct;
```

```
anmp=&dg;
```

```
cout<< typeid(*anmp).name();
```

displays the information of the object pointed by `anm` pointer Similarly

```
Cat *cpt;
```

```
cpt=dynamic_cast<Cat*>(panm);
```

The down cast is successful if `panm` is holding the address of objects of class `Cat`.

Exercises

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.

```

#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 paralleleside[2];
    float nonparallelside[2];
public:
    trapezoid(float a1, float a2, float b1, float
b2, string n= "Trapezoid")
    {
        paralleleside[0] = a1;
        paralleleside[1] = a2;
        nonparallelside[0] = b1;
        nonparallelside[1] = b2;
        sname = n;
    }
    float area()
    {
        sarea =
(paralleleside[0]+paralleleside[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;
}

```

```

#include<iostream>//or
#define PI 3.141592
using namespace std;
class shape
{
public:
    virtual void getdata()=0;
    virtual void area()=0;
    virtual void display()=0;
    virtual ~shape()
    {
        cout<<"Base class destructor called"<<endl;
    }
};
class circle:public shape
{
    float r,a;
public:
    void getdata()
    {
        cout<<endl<<"Enter the radius:";
        cin>>r;
    }
    void area()
    {
        a=PI*r*r;
    }
    void display()
    {
        cout<<"\nArea="<<a<<endl;
    }
}

```

```

        cout<<"Circle"<<endl;
    }
    ~circle()
    {
        cout<<"Circle class destructor called"<<endl;
    }
};
class rectangle:public shape
{
    float l,b,a;
public:
    void getdata()
    {
        cout<<endl<<"Enter length:";
        cin>>l;
        cout<<"\nEnter breadth:";
        cin>>b;
    }
    void area()
    {
        a=l*b;
    }
    void display()
    {
        cout<<"\nArea="<<a<<endl;
        cout<<"Rectangle"<<endl;
    }
    ~rectangle()
    {
        cout<<"Rectangle class destructor
called"<<endl;
    }
};
class trapezoid:public shape
{
    float l1,l2,h,a;
public:
    void getdata()
    {
        cout<<endl<<"Enter the length of first parallel
side:";
        cin>>l1;

```



```

        cout<<"\nEnter the length of the second
parallel side:";
        cin>>l2;
        cout<<"\nEnter the height:";
        cin>>h;
    }
    void area()
    {
        a=h*(l1+l2)/2;
    }
    void display()
    {
        cout<<"Area="<<a<<endl;
        cout<<"Trapezoid"<<endl;
    }
    ~trapezoid()
    {
        cout<<"Trapezoid class destructor
called"<<endl;
    }
};

int main()
{
    shape *bobj;
    bobj=new circle;
    bobj->getdata();
    bobj->area();
    bobj->display();
    delete bobj;
    bobj=new rectangle;
    bobj->getdata();
    bobj->area();
    bobj->display();
    delete bobj;
    bobj=new trapezoid;
    bobj->getdata();
    bobj->area();
    bobj->display();
    delete bobj;
    return 0;
}

```

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.

```
#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 xyz("ABCD","EFGH",50000);
    cout << "name " << xyz.getname() << endl;
    cout << "major " << xyz.getmajor() << endl;
    cout << "salary " << xyz.getsalary() <<
    endl; return 0;
};

#include<iostream>//or
#include<conio.h>
using namespace std;
class Person

```

```

{
    int age;
public:
    void person_getdata()
    {
        cout<<"Enter person's age:";
        cin>>age;
    }
    void person_disp()
    {
        cout<<"\nAge="<<age<<endl;
    }
};
class Employee:virtual public Person
{
    int emp_id;
public:
    void employee_getdata()
    {
        cout<<endl<<"Enter employee id:";
        cin>>emp_id;
    }
    void employee_disp()
    {
        cout<<"\nEmployee id="<<emp_id<<endl;
    }
};
class Student:virtual public Person
{
    int roll_no;
public:
    void student_getdata()
    {
        cout<<endl<<"Enter student's roll number:";
        cin>>roll_no;
    }
    void student_disp()
    {
        cout<<"\nRoll number="<<roll_no<<endl;
    }
};

```

```

class Manager:public Student,public Employee
{
    string company_name;
public:
    void manager_getdata()
    {
        cout<<endl<<"Enter company name:";
        cin>>company_name;
    }
    void manager_disp()
    {
        cout<<"\nCompany name="<<company_name<<endl;
    }
};

int main()
{
    Manager mobj;
    mobj.person_getdata();
    mobj.employee_getdata();
    mobj.student_getdata();
    mobj.manager_getdata();
    system("cls");
    mobj.person_disp();
    mobj.employee_disp();
    mobj.student_disp();
    mobj.manager_disp();
    return 0;
}

```

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

```
#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("Ijkl",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;
}

#include<iostream>//or
using namespace std;
class Student
{
public:
    virtual void getdata()=0;
    virtual void display()=0;
};
class Engineering:public Student
{
    string faculty;
public:
    void getdata()
    {
        cout<<"\nEnter your faculty:"<<endl;
        cin>>faculty;
    }
    void display()
    {
        cout<<"\nYour faculty is "<<faculty<<endl;
    }
};
class Medicine:public Student
{
    int n;
public:
    void getdata()
    {
        cout<<"\nHow many stethoscope do you have?";
        cin>>n;
    }
    void display()
    {
        cout<<"\nNumber of stethoscope = "<<n<<endl;
    }
};
class Science:public Student

```



```

{
    string ans;
public:
    void getdata()
    {
        cout<<"\nDo you love physics?";
        cin>>ans;
    }
    void display()
    {
        cout<<"\nYour answer = "<<ans<<endl;
    }
};

int main()
{
    Engineering eobj;
    Medicine mobj;
    Science sobj;
    Student *s[3];
    s[0]=&eobj;
    s[0]->getdata();
    s[0]->display();
    s[1]=&mobj;
    s[1]->getdata();
    s[1]->display();
    s[2]=&sobj;
    s[2]->getdata();
    s[2]->display();
}

```

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.

```
#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;
}

#include <iostream> //or
#include <cstring>
#include <typeinfo>
using namespace std;
class Vehicle
{
protected :
    string vec_name;
    int no_of_wheels;
public:
    Vehicle(string name, int wheels)
    {
        vec_name=name;
        no_of_wheels=wheels;
    }
    void display()
    {
        cout<<"From Vehicle:"<<endl;
        cout<<"name ="<<vec_name<<endl<<"no of
wheels="<<no_of_wheels<<endl<<endl;
    }
};
class Bus:public Vehicle
{
    int bus_id;
public:
    Bus (string name, int wheels,int
id):Vehicle(name,wheels)
    {
        bus_id=id;
    }
    void display()
    {
        cout<<"From Bus:"<<endl;
        cout<<"name ="<<vec_name<<endl<<"no of
wheels="<<no_of_wheels<<endl<<"bus id
="<<bus_id<<endl<<endl;
    }
}

```

```

};
class Car:public Vehicle
{
    int car_id;
public:
    Car (string name, int wheels,int
id):Vehicle(name,wheels)
    {
        car_id=id;
    }
    void display()
    {
        cout<<"From Car:"<<endl;
        cout<<"name ="<<vec_name<<endl<<"no of
wheels="<<no_of_wheels<<endl<<"car id
="<<car_id<<endl<<endl;
    }
};
class Bike:public Vehicle
{
    int bike_id;
public:
    Bike (string name, int wheels,int
id):Vehicle(name,wheels)
    {
        bike_id=id;
    }
    void display()
    {
        cout<<"From Bike:"<<endl;
        cout<<"name ="<<vec_name<<endl<<"no of
wheels="<<no_of_wheels<<endl<<"bike id
="<<bike_id<<endl<<endl;
    }
};
int main()
{
    Vehicle *v[3];
    Bus *bus =new Bus("Tata",6,123);
    Car *car = new Car("Tesla",4,342);
    Bike *bike = new Bike("ninja",2,898);

```

```
    cout<<"Before type casting:"<<endl;
    bus->display();
    car->display();
    bike->display();
    v[0]=dynamic_cast<Vehicle *>(bus);
    v[1]=dynamic_cast<Vehicle *>(car);
    v[2]=dynamic_cast<Vehicle *>(bike);
    cout<<"After type casting:"<<endl;
    for(int i=0;i<3;i++)
    {
        v[i]->display();
        cout<<endl;
    }
    cout<<"Type ID:"<<endl;
    cout<<typeid(*bus).name()<<endl;
    cout<<typeid(*car).name()<<endl;
    cout<<typeid(*bike).name()<<endl;
}
```