Experiment No: 6 Date:

Aim: To study virtual function and Polymorphism (run time polymorphism)

Theory:

1. Introduction
   - Virtual functions enable dynamic polymorphism in C++.
   - Achieved through a base class pointer pointing to derived class objects.
2. Virtual Functions:
   - Declared in the base class with the `virtual` keyword.
   - Overridden in derived classes to provide specific implementations.
   - Resolve at runtime based on the actual type of the object.
3. Polymorphism:
   - Enables objects of different types to be treated as objects of a common base type.
   - Resolves function calls at runtime based on the actual type of the object.
4. Pure Virtual Functions:
   - Functions declared with `virtual` and assigned `0` are pure virtual.
   - Abstract base classes contain at least one pure virtual function.
   - Must be overridden by derived classes.
5. Abstract Classes:
   - Classes containing pure virtual functions are abstract.
   - Cannot be instantiated; used as base classes.
   - Derived classes must provide implementations for pure virtual functions.
6. Dynamic Binding:
   - Occurs during runtime.
   - Function calls resolved based on the actual type of the object.
7. Static Binding:
   - Occurs during compile-time.
   - Function calls resolved based on the declared type of the pointer or reference.
8. Advantages:
   - Enables code extensibility.
   - Facilitates code maintenance and scalability.
   - Supports the creation of frameworks and libraries.

[A] Write a C++ program to understand virtual functions in C++

```
#include<iostream>
using namespace std;

class base{
public:
   void display(){
      cout<<"Display Base"<<endl;
   }
   virtual void show(){
      cout<<"Show Base"<<endl;
   }
};
class derived: public base{
public:
   void display(){
      cout<<"Display Derived"<<endl;
   }
   void show(){
      cout<<"Show Derived"<<endl;
   }
};

int main(){
   base B;
   derived D;
   base *ptr = &B;
   ptr->display();
```

```
    ptr->show();
    ptr = &D;
    ptr->display();
    ptr->show();
    return 0;
}
```

Output:

```
Display Base
Show Base
Display Base
Show Derived
```

[B] Write a C++ program to understand pure virtual functions in C++

```
#include<iostream>
#include <cmath>
using namespace std;

#define PI 3.14

class Shape {
public:
    virtual float area() = 0;

    virtual float perimeter() = 0;
};

class Circle : public Shape {
private:
    float radius;
public:
    Circle(float r) : radius(r) {}

    float area(){
        return PI * radius * radius;
    }
    float perimeter(){
        return 2 * PI * radius;
    }
};

class Rectangle : public Shape {
private:
    float length, width;

public:
    Rectangle(float l, float w) : length(l), width(w) {}

    float area() {
        return length * width;
    }

    float perimeter() {
        return 2 * (length + width);
    }
};

int main() {
    Shape *ptr;
    Circle circle(5);
    Rectangle rectangle(4, 6);
    ptr = &circle;
    cout << "Circle Area: " << ptr->area() << ",
Perimeter: " << ptr->perimeter() << endl;
    ptr = &rectangle;
    cout << "Rectangle Area: " << ptr->area() << ",
Perimeter: " << ptr->perimeter() << endl;

    return 0;
}
```

Output:

```
Circle Area: 78.5, Perimeter: 31.4
Rectangle Area: 24, Perimeter: 20
```

Conclusion: the concepts of virtual function and polymorphism were understood and implemented in the codes above.

Nitesh Naik

(Subject Faculty)