EXPERIMENT NO - 15

Title: Write a program to implement Virtual function and Pure virtual function

Objectives:

1. To understand the concept of Virtual function and Pure virtual function

Key Concepts: Virtual function and Pure virtual function

Theory:

Virtual Function:

A virtual function is a member function that is declared in the base class using the keyword virtual and is re-defined (Overriden) in the derived class. It tells the compiler to perform late binding where the compiler matches the object with the right called function and executes it during the runtime. This technique of falls under Runtime Polymorphism.

The term Polymorphism means the ability to take many forms. It occurs if there is a hierarchy of classes that are all related to each other by inheritance. In simple words, when we break down Polymorphism into 'Poly – Many' and 'morphism – Forms' it means showing different characteristics in different situations.

Consider the following simple program as an example of runtime polymorphism. The main thing to note about the program is that the derived class's function is called using a base class pointer. The idea is that virtual functions are called according to the type of the object instance pointed to or referenced, not according to the type of the pointer or reference.

In other words, virtual functions are resolved late, at runtime.

C++ Program to Calculate the Area of Shapes using Virtual Function.

```
#include <iostream>
 using namespace std;
 class Shape {
 protected:
  double x, y;
```

```
public:
 void set_dim(double i, double j=0) {
  x = i;
  y = j;
 }
 virtual void show_area(void) {
  cout << "No area computation defined ";
  cout << "for this class.\n";
 }
} ;

class triangle : public Shape {
 public:
  void show_area(void) {
   cout << "Triangle with height ";
   cout << x << " and base " << y;
   cout << " has an area of ";
   cout << x * 0.5 * y << ".\n";
  }
};

class square : public Shape {
 public:
  void show_area(void) {
   cout << "Square with dimensions ";
   cout << x << "x" << y;
   cout << " has an area of ";
   cout << x *  y << ".\n";
```

```
    }
};

class circle : public Shape {
  public:
    void show_area(void) {
      cout << "Circle with radius ";
      cout << x;
      cout << " has an area of ";
      cout << 3.14 * x * x;
    }
} ;

main(void)
{
  Shape *p;
  triangle t;
  square s;
  circle c;

  p = &t;
  p->set_dim(10.0, 5.0);
  p->show_area();

  p = &s;
  p->set_dim(10.0, 5.0);
  p->show_area();
```

```
  p = &c;
  p->set_dim(9.0);
  p->show_area();

  return 0;
}
```

Output of Program

Triangle with height 10 and base 5 has an area of 25.

Square with dimensions 10x5 has an area of 50.

Circle with radius 9 has an area of 254.34

Pure Virtual Functions and Abstract Classes in C++

Pure virtual Functions are virtual functions with no definition. They start with virtual keyword and ends with = 0. Here is the syntax for a pure virtual function,

virtual void f() = 0;

An abstract class is a class in C++ which has at least one pure virtual function.

Abstract class can have normal functions and variables along with a pure virtual function.

Abstract class cannot be instantiated, but pointers and references of Abstract class type can be created.

Abstract classes are mainly used for Upcasting, so that its derived classes can use its interface.

If an Abstract Class has derived class, they must implement all pure virtual functions, or else they will become Abstract too.

```
#include <iostream>

class Shape
```

```
{
public:
   Shape(){}
   virtual ~Shape(){}
   virtual long GetArea() = 0;
   virtual long GetPerim()= 0;
   virtual void Draw() = 0;
private:
};
 void Shape::Draw()
{
   std::cout << "Abstract drawing mechanism!\n";
}
class Circle : public Shape
{
public:
   Circle(int radius):itsRadius(radius){}
   ~Circle(){}
   long GetArea() { return 3 * itsRadius * itsRadius; }
   long GetPerim() { return 9 * itsRadius; }
   void Draw();
private:
   int itsRadius;
   int itsCircumference;
};
 void Circle::Draw()
{
   std::cout << "Circle drawing routine here!\n";
```

```
    Shape::Draw();
}

class Rectangle : public Shape
{
public:
    Rectangle(int len, int width):
        itsLength(len), itsWidth(width){}
    virtual ~Rectangle(){}
    long GetArea() { return itsLength * itsWidth; }
    long GetPerim() {return 2*itsLength + 2*itsWidth; }
    virtual int GetLength() { return itsLength; }
    virtual int GetWidth() { return itsWidth; }
    void Draw();
private:
    int itsWidth;
    int itsLength;
};

void Rectangle::Draw()
{
    for (int i = 0; i<itsLength; i++)
    {
        for (int j = 0; j<itsWidth; j++)
            std::cout << "x ";

        std::cout << "\n";
    }
```

```
    Shape::Draw();
}

class Square : public Rectangle
{
public:
    Square(int len);
    Square(int len, int width);
    ~Square(){}
    long GetPerim() {return 4 * GetLength();}
};

Square::Square(int len):Rectangle(len,len)
{}

Square::Square(int len, int width):Rectangle(len,width){
    if (GetLength() != GetWidth())
        std::cout << "Error, not a square... a Rectangle??\n";
}

int main()
{
    Shape * sp;

    sp = new Circle(5);
    sp->Draw();

    sp = new Rectangle(4,6);
```

```
    sp->Draw();

    sp = new Square (5);
    sp->Draw();

    return 0;
}
```

Output:

```
Circle drawing routine here!
Abstract drawing mechanism!
x x x x x x
x x x x x x
x x x x x x
x x x x x x
Abstract drawing mechanism!
x x x x x
x x x x x
x x x x x
x x x x x
x x x x x
Abstract drawing mechanism!
```

Problem Statement:

Write program to implement pure virtual function

1. Create Base class with name Employee
2. Create Derived classes Engineer, TeamLeader and Manager derived from class Employee with data member as salary
3. Create a pure virtual function raiseSalary() in Employee
4. Implement function raiseSalary() in classes Engineer, TeamLeader and Manager to increase the salary as per given percentage
5. Create array of pointers of size 3 to Employee as Employee *e[3]
6. Assign e[0] with address of Engineer object, assign e[1] with address of TeamLeader object, assign e[2] with address of Manager object
7. Call e[i]->raiseSalary() in loop to raise the salary of Engineer, TeamLeader and Manager