National University of Computer and Emerging Sciences

Lab Manual # 01

Data Structures

OOP Concepts and Pointers

Lab Instructor Ms. Sukhan Amir
Semester Fall 2024

Department of Computer Science
FAST-NU, Lahore, Pakistan

Objective:

After performing this lab, below mentioned concepts would be revised

1. Dynamic Allocation
2. Composition, Inheritance, Polymorphism

3. Pointers

Instructions:

• Make a separate project for each task.
• Indent your code properly.
• Use meaningful variable and function names. Follow the naming conventions. • Use meaningful prompt lines and labels for all input/output. • Make sure that there are NO dangling pointers or memory leaks in your program.

Task 1: Dynamic Allocation (3+3.5+3.5)

A ragged array is an array which contains a varying number of elements in each row. The Pascal triangle can be used to compute the coefficients of the terms in the expansion of (a + b) n. Write a function that creates a ragged array representing the Pascal triangle. In a Pascal triangle, each element is the sum of the element directly above it (if any) and the element to the left of the element directly above it (if any).

A Pascal triangle of size 7 is shown below:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

You are required to define the following functions:

Task 1.1:

int createPascalTriangle (int n);**

This function will take an integer (n) as argument and create a Pascal triangle consisting of n rows. It will dynamically allocate the two-dimensional ragged array, fill up its elements, and return a pointer to this filled array (Pascal triangle).

Task 1.2:

void displayPascalTriangle (int pt, int n);**

This function will take a pointer pt which is pointing to a Pascal triangle consisting of n rows. It

will display the Pascal triangle on screen.

Task 1.3:

void deallocatePascalTriangle (int pt, int n);**

This function will take a pointer pt which is pointing to a Pascal triangle consisting of n rows. This function will deallocate the two-dimensional array containing the Pascal triangle.

Write a main function which asks the user to specify the value of n. After that the main should call the above functions to create a Pascal triangle of size n, display it on screen, and, finally, de allocate all the dynamically allocated memory.

Task 2: (5+5)

Task 2.1:

Implement a class named **Employee**. The class should keep the following information in private member variables:

- Employee name (a cstring which can contain at most 50 characters)
- Employee number (an integer)
- Hire date (a cstring which can contain at most 20 characters)

Implement the parameterized constructor and the appropriate getter and setter functions for the **Employee** class.

Next, implement a class named **ProductionWorker** that is derived from the **Employee** class. The **ProductionWorker** class should have member variables to hold the following information:

- Shift (an integer)
- Hourly pay rate (a double)

The workday is divided into two shifts: day and night. The shift variable will hold an integer value representing the shift that the employee works. The day shift is shift 1 and the night shift is shift 2. Implement a parameterized constructor for the **ProductionWorker** class which should take the initial values of all attributes. Also implement the appropriate getter and setter functions. Demonstrate the classes by writing a program that uses a **ProductionWorker** object.

Task 2.2:

In a particular factory a shift supervisor is a salaried employee who supervises a shift. In addition to a salary, the shift supervisor earns a yearly bonus when his or her shift meets production goals. Implement a **ShiftSupervisor** class that is derived from the **Employee** class you created in Task 2.1.

The **ShiftSupervisor** class should have a member variable that holds the **annual salary** (a double) and a member variable that holds the **annual production bonus** (a double) that a shift supervisor has earned. Write a parameterized constructor and the appropriate getter and setter functions for the class. Demonstrate the class by writing a program that uses a **ShiftSupervisor** object.

Task 3: Operator Overloading and Polymorphism (4+4+2)

Polymorphism is the ability (in programming) to present the same interface for differing underlying forms (data types). For example, in many languages, integers and floats are implicitly polymorphic since you can add, subtract, multiply and so on, irrespective of the fact that the types are different. They're rarely considered as objects in the usual term. But, in that same way, a class like BigDecimal or Rational or Imaginary can also provide those operations, even though they operate on different data types.

Now your task is to create a **base class Calculator** and add 4 pure virtual function **add, subtract, multiply and divide** in it. Now create 2 child **class SimpleNumbers** and **ComplexNumbers** and add 4 function add, subtract, multiply and divide in them as well. Structure of classes are:

```
Class SimpleNumber
{
        int a;
        int b;
 public:
        void add();
        void subtract();
        void multiply();
        void divide();
```



```
}
```

```
Class ComplexNumber
{
        ComplexNum a;
 ComplexNum b;
public:
        void add();
        void subtract();
        void multiply();
        void divide();
}
//create a struct for ComplexNum
class ComplexNum
{
        int real;
         int img;
}
```

For Complex Numbers the Formulas are:

- (a + bi) + (c + di) = (a + c) + (b + d)i **(for addition)**
- (a + bi) - (c + di) = (a - c) + (b - d)i **(for subtraction)**
- (a + bi)(c + di) = (ac – bd) + (bc + ad)i **(for multiplication)**
- (a + bi) / (c + di) = (ac + bd / c^2 + d^2) + (bc – ad / c^2 + d^2) **(for division)**

Demonstrate the classes in a program that using **Calculator** pointers. Take input from User that whether you want to create the object of **ComplexNumber** or **SimpleNumber** (dynamically allocated) and take values for that object from user. This Dynamic object should be pointed by **Calculator** Class. Then by using Calculator pointer, call object's all functions (add, subtract, multiply and divide) which decides on Runtime that which object's functions it call i.e. **ComplexNumber** or **SimpleNumber**.

Task 4: Operator Overloading

A polynomial P1(x) = x4+2x2+5 has three terms: x4 2x2 and 5. Coefficients of these terms are 1, 2 and 5 respectively while exponents are 4, 2 and 0 respectively. To work with Polynomials, a
definition of class Polynomial is given below and memory configuration for P1 is shown as

follows:

Your task is to complete the definition of Polynomial class such that the main program runs

successfully. Make sure that your program doesn't consume extra memory space and it should not

leak any memory.

```
void main()

{

int coeff_P1[] = {1,2,5}; //Coefficients for Polynomial P1

int exp_P1[] = {4,2,0}; //Exponents for Polynomial P1

int coeff_P2[] = {4,3}; //Coefficients for Polynomial P2

int exp_P2[] = {6,2}; //Exponents for Polynomial P2

Polynomial P1(3, coeff_P1, exp_P1);//Creates P1 with 3 terms (P1 = 1x^4 + 2x^2 +

5x^0 )

Polynomial P2(2, coeff_P2, exp_P2);//Creates P2 with 2 terms (P2 = 4x^6 + 3x^2)

cout<<"P1 = "<<P1<<endl; //Prints P1 = x^4+2x^2+5

cout<<"P2 = "<<P2<<endl; //Prints P2 = 4x^6+3x^2

if(!P1)
cout<<"P1 is zero"<<endl; /*if the polynomial has only 1 term and its coeff and exp are zero.
```

i.e.

```
if p1 = 0.*/

if(P1 != P2)

cout<<”P1 is Not Equal to P2”<<endl;

cout<<++P1<<endl; //adds 1 in all the coefficients.

cout<<P1<<endl;

cout<<P1++<<endl; //adds 1 in all the coefficients.

cout<<P1<<endl;

Polynomial P3 = P1+P2; //Adds P1 and P2 and saves result in P3.You may

consume extra space for resultant Polynomial in Add function

cout<<"P3 = "<<P3<<endl; //Prints P3 = 4x^6+x^4+5x^2+5

P3 = P1+2;

cout<<"P3 = "<<P3<<endl
```