Lab 1 Class and Object

1. Create a class called Area as shown below. Implement the member functions and write the driver code (main function) to call an object of the Area class and perform calculation.

```
class Area {
float r, area;
public:
   void input() ; // input value for the data members
   void findArea(); // Calculate area of the circle
   void display() ; // Display area of the circle
};
```

2. Create a class called Student with following data members:

```
int roll_no;
int marks[5];
int age;
Implement the following member functions:
void getdata ();
void tot_marks ();
Your main function looks like that:
void main()
{
student stu;
stu.getdata() ;
stu.tot_marks() ;
}
```

3. Create a class called Fraction as shown below and implement the member functions and write the driver code to test the program.

```
class Fraction
{
private:
   int numerator;          // no restrictions
   int denominator;              // Invariant:  denominator != 0
public:
     void Input();          // input a fraction from keyboard.
```

```
        void Show();                // Display a fraction on screen
        int GetNumerator();
        int GetDenominator();
        void SetValue(int n, int d); // set the fraction's value through
    parameters
  double Evaluate();     // Return the decimal value of a fraction
};
```

4. *Create a class called Employee as shown below. Net Salary of the employee is calculated as (basic+da)-it; Store and show the details of 5 employee.*
   ***Note:** here data members are private, therefore you can't access the basic, da and it . Inside show_emp_details() call fund_net_salary function.*

```
class Employee
{
    int   emp_number;
    char  emp_name[20];
    float emp_basic;
    float emp_da;
    float emp_it;
    public:
        void get_emp_details();
        float find_net_salary(float basic, float da, float it);
        void show_emp_details();
};
```

5. Create a class called DList as shown below. Implement the member functions and write the driver code (main function) to call an object of the DList class and perform calculation.

```
const int MAX = 10;

class DList
{
private:
  double array[MAX];
  int current;          // number of stored items (max is 10)
```

```
public:
  bool Insert(double item);     // inserts item into list (if room)
  double GetElement(unsigned int n);          // returns element at
index n
  void Print();             // prints the list
  int GetSize();                    // returns number of elements in list
};
```

6. Modify above code by adding the following member functions to the DList class.

```
bool Delete(int n);  // delete the nth element of the list
        //  return true for success, false if n not a valid position

 double Sum();        // return the sum of the elements of the list
double Average();    // return the average of the elements of the list
  double Max();        // return the maximum value in the list
  void Clear();        // reset the list to empty
  int Greater(double x); /* count how many values in the list are
greater  than x.  Return the count.*/
```

Lab 2

1. Create a class called Complex for performing arithmetic with complex numbers. Write a driver program to test your class.
   Complex numbers have the form
   realPart + imaginaryPart * i
   where *i* is $\sqrt{-1}$

   Use double variables to represent the private data of the class. Provide a constructor function that enables an object of this class to be initialized when it is declared. The constructor should contain default values in case no initializers are provided. Provide public member functions for each of the following:
   a) Addition of two Complex numbers: The real parts are added together and the imaginary parts are added together.
   b) Set complex number is used to assign value to real and imaginary part.
   c) Subtraction of two Complex numbers: The real part of the right operand is subtracted from the real part of the left operand and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.
   d) Printing Complex numbers in the form (a, b) where a is the real part and b is the imaginary part.

   Driver Code: int main()
   {
   Complex b(1, 7), c(9, 2);
   b.printComplex();
   cout << " + ";
   c.printComplex();
   cout << " = ";
   b.addition(c);
   b.printComplex();

   cout << '\n';

   b.setComplexNumber(10, 1); // reset realPart and imaginaryPart

   c.setComplexNumber(11, 5);
   b.printComplex();
   cout << " - ";
   c.printComplex();
   cout << " = ";
   b.subtraction(c);
   b.printComplex();

```
 cout << endl;
 return 0;}
```

2. Create a class Rectangle. The class has attributes length and width, each of which defaults to 1. It has member functions that calculate the perimeter and the area of the rectangle. It has *set* and *get* functions for both length and width. The *set* functions should verify that length and width are each floating-point number larger than 0.0 and less than 20.0.

   Driver Code: int main()

```
{
Rectangle a, b( 4.0, 5.0 ), c( 67.0, 888.0 );
// output Rectangle a
 cout << "a: length = " << a.getLength()<< "; width = " << a.getWidth()<< ";
perimeter = " << a.perimeter() << "; area = " << a.area() << '\n';

 // output Rectangle b
cout << "b: length = " << b.getLength() << "; width = " << b.getWidth()<< ";
perimeter = " << b.perimeter() << "; area = " << b.area() << '\n';

// output Rectangle c;
 cout << "c: length = " << c.getLength()<< "; width = " << c.getWidth()<< ";
perimeter = " << c.perimeter() << "; area = “ << c.area() << endl;
 return 0;
}
```

3. Modify the above Rectangle class. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* function that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single *x* or *y* coordinate larger than 20.0. The *set* function also verifies that the supplied coordinates do, in fact, specify a rectangle. Member functions calculate the length, width, perimeter and area. The length is the larger of the two dimensions. Include a predicate function IsSquare that determines if the rectangle is a square.

   Driver Code: int main()

```
{
double w[ 2 ] = { 1.0, 1.0 }, x[ 2 ] = { 5.0, 1.0 },
y[ 2 ] = { 5.0, 3.0 }, z[ 2 ] = { 1.0, 3.0 },
```

```
 j[ 2 ] = { 0.0, 0.0 }, k[ 2 ] = { 1.0, 0.0 },
 m[ 2 ] = { 1.0, 1.0 }, n[ 2 ] = { 0.0, 1.0 },
 v[ 2 ] = { 99.0, -2.3 };
 Rectangle a( z, y, x, w ), b( j, k, m, n ), c( w, x, m, n ), d( v, x, y, z );

//output length, area, perimeter and IsSquare.
return 0;
 }
```

4. Create a class TicTacToe that will enable you to write a complete program to play the game of tic-tac-toe. The class contains as private data a 3-by-3 double array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square; place a 2 wherever the second player moves. Each move must be to an empty square. After each move, determine if the game has been won or if the game is a draw.

5. Create a class SimpleLinkedList, this class will use the structure 'node' for the creation of the linked list. Node is structure declare globally and class define the private node variables and public member functions as shown below. Add nodes and display them through driver code.

```
struct node
{
   int data;
   node *next;
};

class SimpleLinkedList
{
private:
   node *head;
public:
// member functions
};
```

Use constructor to initialize head to null and member functions are:

```
 void add_node(int n)
{
```

```
// To add elements in the linked list.

}


Void display_linkedlist()
{
// show the content of the linked list
}
```

Lab 4

Instruction: For each and every assignment, create a project. Define the class, data members and member functions in .h file. Implementation detail in separate file and driver code in separate file. In the driver code create an object and perform operations to test the utility of the project.

1. Create a class called Stack, Implement Stack using an array.

```
class stack
{
   int *arr;
   int top;
   int capacity;
public:
   stack(int size = SIZE);        // constructor
   ~stack();                      // destructor
    void push(int); // push the element into the stack
   int pop(); //  pop the element from the top
   int peek(); // search the element in the stack
   int size(); // Return size of stack
   bool isEmpty(); // return true if stack is empty
   bool isFull(); // return true if stack is full
};
```

2. Create a class called stack similar member functions as above. Implement stack using linked list.
3. Modify question 1, design a stack that returns the minimum element in constant time. Use two stacks, main stack to store elements and auxiliary stack to store the required elements needed to determine the minimum number in constant time. Other operations on the stack remain same (push, pop, peek). If the element is removed from the main stack, similar element should also be removed from the auxiliary stack if present.
4. Optimize the above code to return the minimum element in constant time without using auxiliary stack.
5. Implement two stacks in a single array. Both the stacks can grow towards each other with no fixed capacity (array is not divided into two halves, one for each stack).

```
class Stack
{
   int *arr;
   int capacity;
```

```
    int top1, top2;
public:
    // Constructor
    Stack(int n)
    {
        capacity = n;
        arr = new int[n];
        top1 = -1;
        top2 = n;
    }
 void push1(int key); // Function to insert a given element into the first stack
 void push2(int key);  // Function to insert a given element into the second stack
 int pop1(); // Function to pop an element from the first stack
 int pop2(); // Function to pop an element from the second stack
 int size1(); // Return size of first stack
 int size2(); // Return size of second stack
};
```

6. Update the assignment 1 with additional member function called **sortstack**. This function should be a recursive method to sort a stack in ascending order.
7. Write a program to convert infix expression into postfix and prefix expression using the class Stack. Stack is of type char.

```
int main()
{
stack s1;
s1.getInfixExpr(); // function which store expression, "(P+(Q*R)/(S-t))"
 s1.infixToPostfix(); // convert infix into postfix
s1.infixToPrefix(); // convert infix to prefix
   return 0;
}
```

Lab 5: Queue, Stack, Operator Overloading

1. Create a class called IntQueue, implement integer queue using an array and linked list. Implement the following functions:
   create: Create a new, empty queue object.
   empty: Determine whether the queue is empty; return true if it is and false if it is not.
   enqueue: Add a new element at the rear of a queue.
   dequeue: Remove an element from the front of the queue and return it. (This operation cannot be performed if the queue is empty.)
   front: Return the element at the front of the queue (without removing it from the queue). (Again, this operation cannot be performed if the queue is empty.)
   Example:
   Input: Enter your choice
   Press 1. To create a queue
   Press 2. To check whether queue is empty or not
   Press 3. Add new element to queue
   Press 4. Remove element from queue
   Press 5. Show the element at front
   Press 6. To exit
   1
   Queue is created
   Enter your choice again
   3
   Enter new element
   4
   ….
2. In the last lab, we implemented the integer stack. Now we have integer stack and integer queue. Our task is to compute an *alternating series* using the numbers that you entered. An alternating series switches the sign of the number being added for every other number. For example, if we enter the number 1, 3, 15, 9 then we would compute the alternating series as 1 - 3 + 15 - 9 = 4. Modify the intQueue code so that it computes the sum of an alternating series of the numbers as they are removed from the queue (so the first number is positive, the second is negative, etc.). Display the alternating series to the screen on a single line. So if the numbers 1, 3, 15, and 9 were entered by the user, the code should display:
   1 - 3 + 15 - 9 = 4.

Next modify the integer stack code so that it computes the sum of an alternating series of the numbers as they are removed from the stack (so the first number off the stack is positive, the second is negative, etc.). Display the alternating series to the screen on a single line as above. How is the result off the stack different from the result off the queue. In the comments section at the top of your code, explain how this is different than the result you get when using a queue. In addition, there are some cases where the result of using either a stack or a queue will be exactly the same. Indicate in your comment under what conditions the result will be the same. Finally, how could you modify your stack code so that it *always* gives the same result as your queue code? Indicate that in your comments as well.

3. Your next task is to take what we've done with integer queues and stacks and apply it to a new data type. You should make two new classes - stringQueue and stringStack. Using integerQueue and integerStack as a reference, write up two programs that will take a sequence of Strings from the console and place them into either a queue or a stack as appropriate until the user provides a line with ** as the first two characters. Then your code should process the queue (or stack) in the appropriate order, concatenating all of the strings together into a single String and then, when the queue is empty, display that string to the console. Again, consider the differences in the behaviours of the two data structures. When might we want to use a stack for String processing instead of a queue? Put a brief description of the differences between the two pieces of code into the comments of your stringStack code and give an example of when you might need to use a stack.

4. We are given a string and we have to perform some steps. In each step we need to take the first character of the string and put it at the end of the string. We have to find out what will be the string after N steps. Consider the string as a queue. At each step dequeue the character from the front and enqueue it at the end and repeat the process for N times except the final character.
   Example:
   Input String: DAIICT
   Output: TDAIIC

5. Given two queues Q1 and Q2 (objects of class queue). Find out Q3=Q1+Q2 (use operator overloading).

   Example

Q1: 2,5,6,3,8

Q2: 1,3,7,4,3

Q3: 3,8,13,7,11

6. In your integer queue class implement sortqueue function which sorts the element in ascending order. Implement another function called as evenOddMerge, merge 2 pre sorted queues into a new larger sorted queue such that all the evens are in order then all of the odds are in order. You may assume that the function sortqueue is always called before evenOddMerge.
   In the main method of your driver program, create two queues with several items each, print the queues, call the sortqueue method then call the merge method. Then print the original and merged queues.
   Example
   Sorted q1 contains 1, 2, 4, 6 and sorted q2 contains 0, 1, 2, 3, 5
   Call the merge method:
   q3 = evenOddMerge(q1, q2);
   After the call to merge, q3 contains 0, 2, 2, 4, 6, 1, 1, 3, 5

7. Create a class RationalNumber (fractions) with the following capabilities:
   a) Create a constructor that prevents a 0 denominator in a fraction and avoids negative denominators.
   b) Overload the addition, subtraction, multiplication and division operators for this class.
   c) Overload the relational and equality operators for this class.

```
class RationalNumber {
 public:
 RationalNumber( int = 0, int = 1 ); // default constructor
 RationalNumber operator+( const RationalNumber& );
 RationalNumber operator-( const RationalNumber& );
 RationalNumber operator*( const RationalNumber& );
 RationalNumber operator/( RationalNumber& );
 bool operator>( const RationalNumber& );
 bool operator<( const RationalNumber& );
 bool operator>=( const RationalNumber& );
 bool operator<=( const RationalNumber& );
 bool operator==( const RationalNumber& );
 bool operator!=( const RationalNumber& );
```

```
 void printRational( void );
 private:
 int numerator;
 int denominator;
};
```

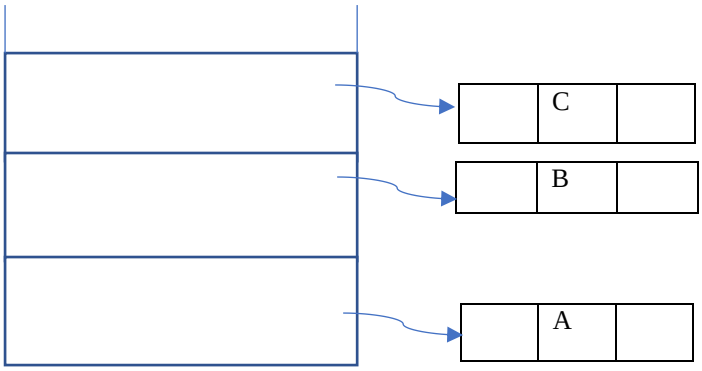
Example:

```
7/3 + 1/3 = 8/3
7/3 - 1/3 = 2
7/3 * 1/3 = 7/9
7/3 / 1/3 = 7
7/3 is:
 > 1/3 according to the overloaded > operator
 >= 1/3 according to the overloaded < operator
 >= 1/3 according to the overloaded >= operator
 > 1/3 according to the overloaded <= operator
 != 1/3 according to the overloaded == operator
 != 1/3 according to the overloaded != operator
```

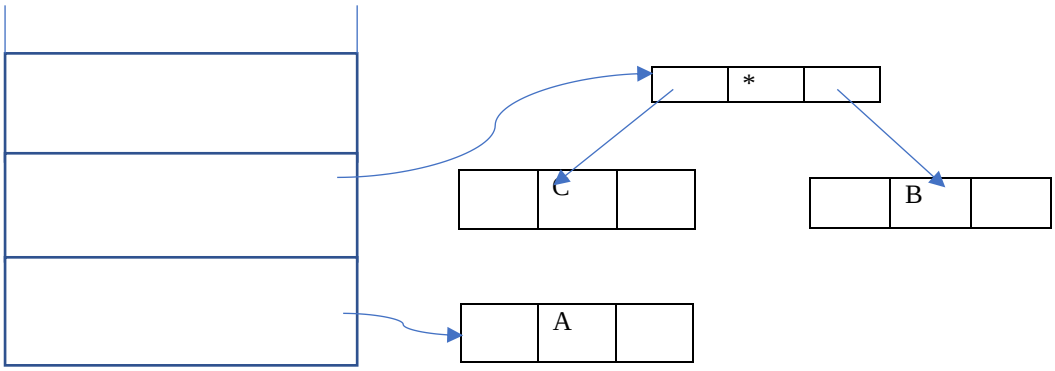
Lab 7

1. A tree representing an expression is called an expression tree. In expression trees, leaf nodes are operands and non-leaf nodes are operators. That means, an expression tree is a binary tree where internal nodes are operators and leaves are operands. An expression tree consists of binary expression. But for a unary operator, one subtree will be empty. The figure below shows a simple expression tree for (A + B * C) / D.

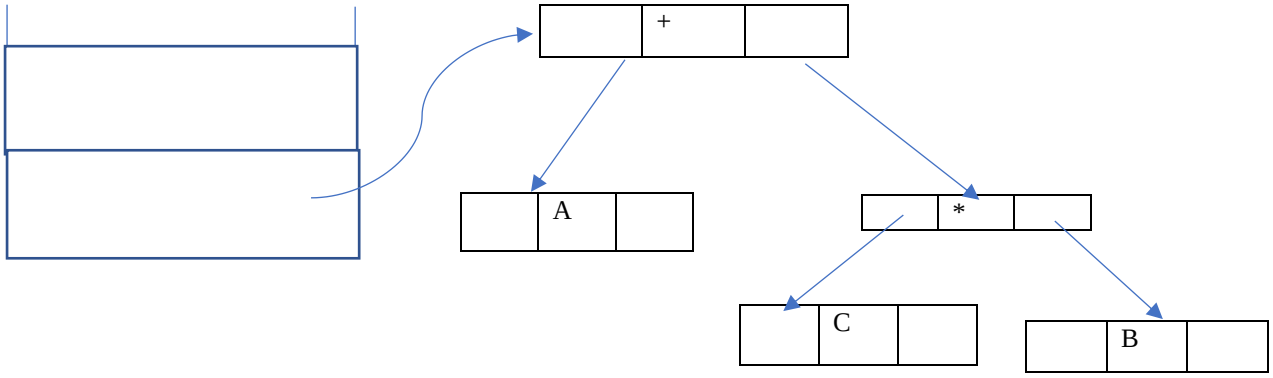
Example : Convert the expression into postfix expression (A B C*+ D/). The first three symbols are operands, so create tree nodes and push pointers to them onto a stack as shown below.



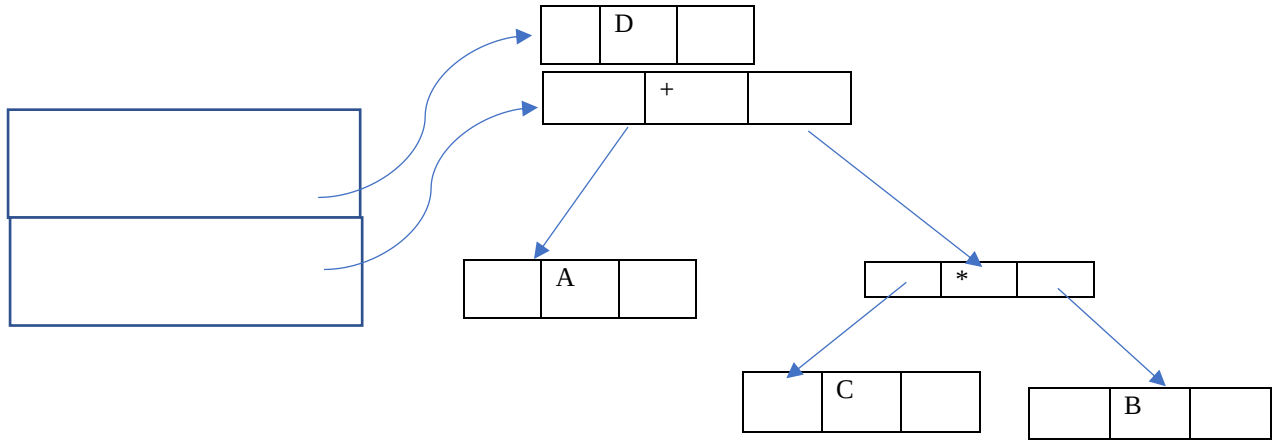
Next, an operator '*' is read, so two pointers to trees are popped, a new tree is formed and a pointer to it is pushed onto the stack.



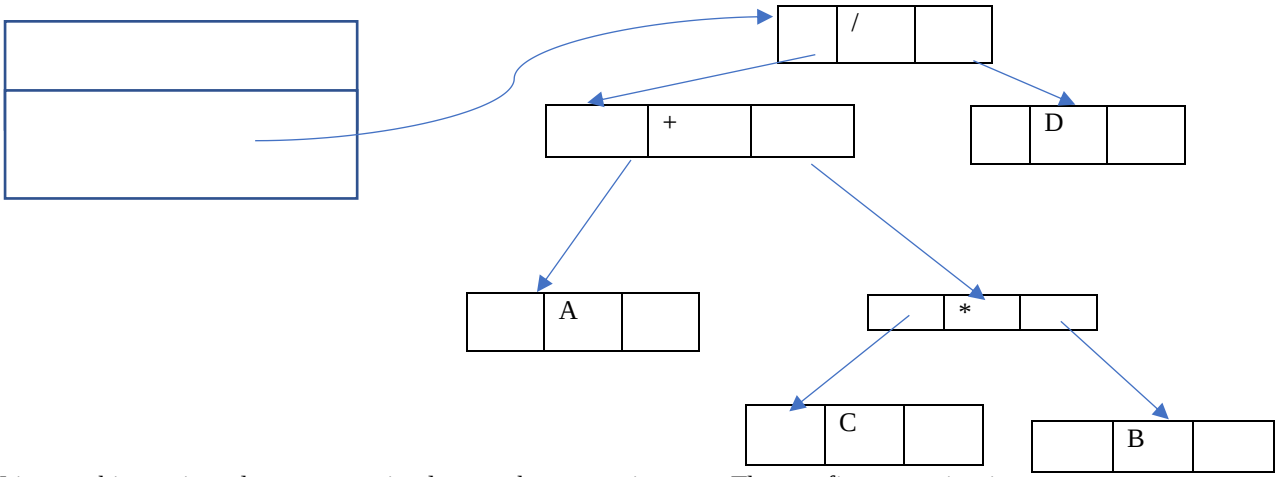
Next, an operator '+' is read, so two pointers to trees are popped, a new tree is formed and a pointer to it is pushed onto the stack.



Next, an operand ‘D’ is read, a one-node tree is created and a pointer to the corresponding tree is pushed onto the stack.



Finally, the last symbol (‘/’) is read, two trees are merged and a pointer to the final tree is left on the stack.

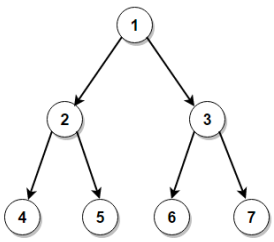


Write an object oriented program to implement the expression tree. The postfix expression is given by user. Implement class and necessary member functions for the same.

2. In the previous lab we implemented the binary tree class and necessary functions. Using the same code perform below tasks:

a. Write a function to find the maximum element in the binary tree and return the number.

b. Write a function to print level order traversal
Example: the level order traversal for below tree is: 4567231



c. Write a function to find the number of half nodes (nodes with only one child) in binary tree.

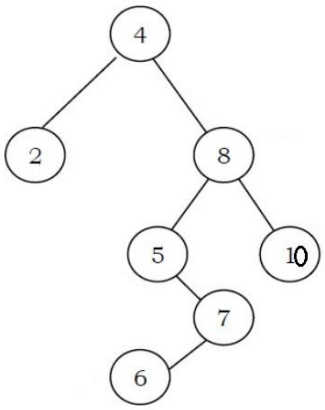
d. Write a function to find the diameter of the binary tree. The diameter of a tree also called as width is the number of nodes on the longest path between two leaves in the tree.

e. Write a function to find the sum of all the elements of the tree.

3. Implement the class called BST (binary search tree). The left sub tree carries nodes smaller than the root node and right sub trees carries nodes larger than the root nodes. This class has following functions: insert, delete, and display.

Implement two programs based on BST class.

a. Write a program to find the shortest path between two given nodes in a BST.
Example: The shortest path between 2 and 5 is 3.



b. Write a program to convert the BST into doubly linked list.
Hint: The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be the same as in Inorder for the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.
Example: