NAME: KAMRAN ANSARI

REG NO: 22MCA0223

Q1. Write an algorithm to convert an infix expression into postfix and prefix expressions using stack. Convert the following expression using the stack.

a) A^B*C-D+E/F/(G+H)

b) A-B/(C*D^E)

Ans → Algorithm for converting infix to postfix

1. Scan the infix expression from left to right.
2. If the scanned character is operand add it to result expression.
3. Else,
   i) If the precedence and associativity of the scanned operator is greater than that of the operator on the stack (or stack is empty / top of stack is '('), then push it to stack.
   ii) Else, pop all the operators from the stack which are of greater than or equal to in precedence than that of scanned operator. After that push the scanned operator to stack.
   (If you encounter parantheis while popping then stop popping and push the scanned operator to stack). All the popped operators go into the result expression.

4. If the scanned character is an '(', push it into the stack.
5. If the scanned character is an ')', pop and add to the expression until a '(' is encountered, discard both the parentheses.
6. Repeat steps 2-5 till the infix expression is scanned.
7. Pop out all remaining operators from the stack and add them to result expression.

Algorithm for converting Infix to Prefix

1. Reverse the infix expression. Note while reversing you must interchange left and right parantheses.
2. Apply the infix to postfix algorithm and obtain the equivalent postfix algorithm.
3. Reverse the postfix expression to get the prefix expression.

a7 A^B*C-D+E/F/(G+H)

Symbol	Stack	Postfix Expression
A		A
^	^	A
B	^	AB
*	*	AB^
C	*	AB^C
-	-	AB^C*
D	-	AB^C*D
+	+	AB^C*D-
E	+	AB^C*D-E
/	+/	AB^C*D-E
F	+/	AB^C*D-EF
/	+/	AB^C*D-EF/
G	+/(	AB^C*D-EF/G
+	+/(+	AB^C*D-EF/G
H	+/(+	AB^C*D-EF/GH
)	+/	AB^C*D-EF/GH+
		AB^C*D-EF/GH+/+

Postfix Expression → AB^C*D-EF/GH+/+

Infix to Prefix

Reverse expression → (H+G)/F/E+D-C*B^A

Applying postfix algorithm to reversed expression.

Symbol	Stack	Postfix Expression
(	(	
H	(	H
+	(+	H
G	(+	HG
)		HG+
/	/	HG+
F	/	HG+F
/	/	HG+F/
E	/	HG+F/E
+	*	HG+F/E/
D	+	HG+F/E/D
-	-	HG+F/E/D+
C	-	HG+F/E/D+C
*	-*	HG+F/E/D+C
B	-*	HG+F/E/D+CB
^	-*^	HG+F/E/D+CB

A	- *^	HG+FIE/D+CBA
		HG+FIE/D+CBA^*-

Postfix of reversed expression →

HG+FIE/D+CBA^*-

Required prefix expression →

-*^ABC+D/E/F+GH

b) A-B/(C*D^E)

Infix to Postfix

Symbol	Stack	Postfix Expression
A		A
-	-	A
B	-	AB
/	-/	AB
(	-/(	AB
C	-/(	ABC
*	-/(*	ABC
D	-/(*	ABCD
^	-/(*^	ABCD
E	-/(*^	ABCDE
)	-/	ABCDE^*
		ABCDE^*/-

Postfix Expression → ABCDE^*/-

Infix to Prefix

Reverse expression → (E^D*C)/B-A

Symbol	Stack	Postfix Expression
(	(	
E	(	E
^	(^	E
D	(^	ED
*	(*	ED^
C	(*	ED^C
)		ED^C*
/	/	ED^C*
B	/	ED^C*B
-	-	ED^C*B/
A	-	ED^C*B/A
		ED^C*B/A-

Postfix of reversed expression →
ED^C*B/A-

Required prefix expression →
-A/B*C^DE

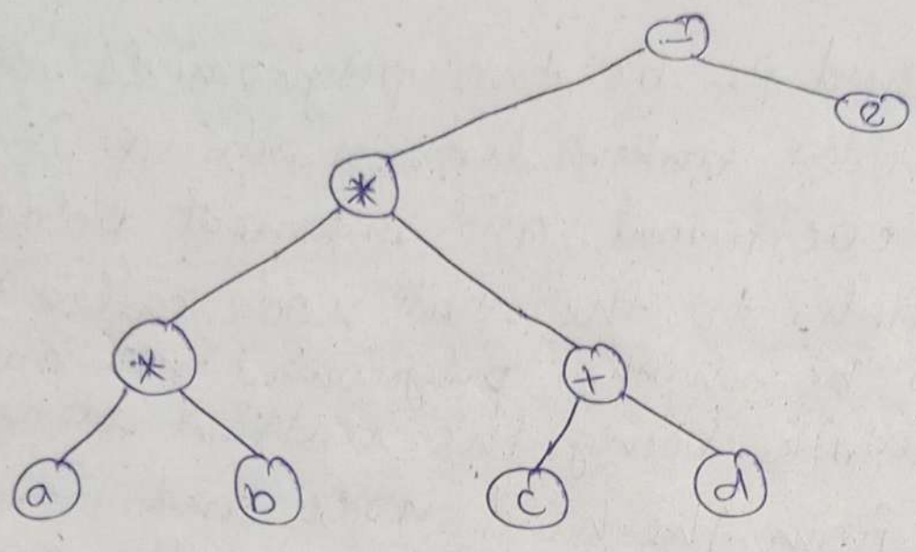
Q.2. Write an algorithm to add two polynomials represented as circular LLA with header node.

Ans →

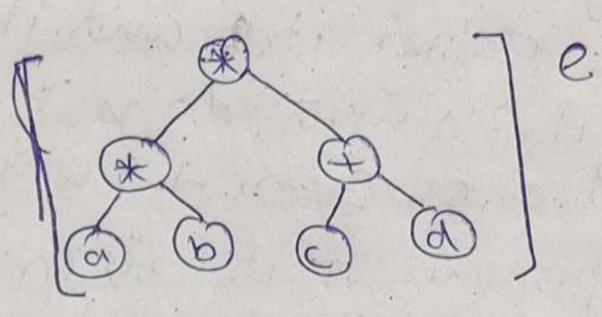
Let p1 and p2 be two polynomials represented as a circular linked lists, made up of nodes having coefficient and exponent fields along with pointer to the next node called next. And p3 be another polynomial with such characteristic, being the resultant array.

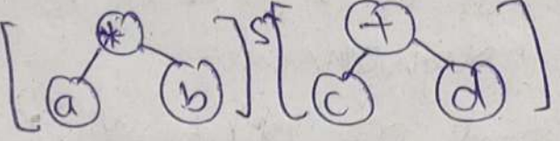
1. Start from the head node and iterate through both p1 and p2 simultaneously using iterators itr1 and itr2, till itr1's next and itr2's next do not point to head, repeat steps 2 to 4 on the pair of iterators
2. If exponent of itr1 is greater than the exponent of itr2, add node itr1 to the resultant polynomial.
3. If exponent of itr2 is greater than the exponent of itr1, add node itr2 to the resultant polynomial.
4. If exponent of itr1 is equal to exponent of itr2 then add a node having coeff. equal to sum of coefficients of itr1 and itr2 and the exponent equal to exponent of itr1 or itr2.
5. While itr1's next is not equal to head, iterate and keep adding itr1 to p3, the resultant polynomial.
6. While itr2's next is not equal to head, iterate and keep adding itr2 to p3, the resultant polynomial.

Q.3. Give the prefix, infix and postfix expressions corresponding to the given tree below.



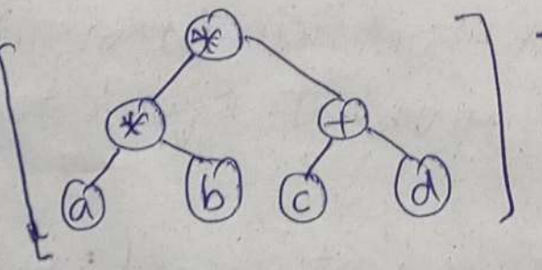
Ans → Prefix

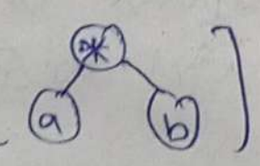
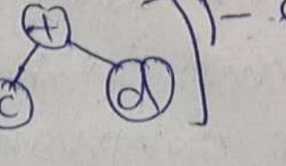
→ - solve  e

→ - * s  e

→ - * * ab + cd e

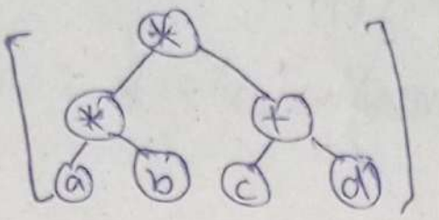
Infix

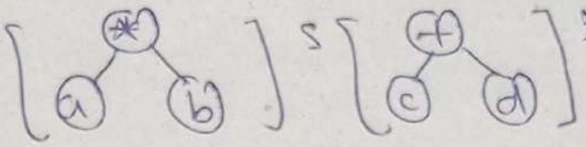
→ s  - e

→ (s  * s ) - e

→ ((a * b) * (c + d)) - e

Postfix

→ 

→ 

→ a b * c d + * e -

∴ Prefix → - * * a b + c d e
Infix → ((a * b) * (c + d)) - e
Postfix → a b * c d + * e -

Q. 4. Create a program that provides a binary converter by using stack. Use stack to store the remainder. The program would get as input a decimal number and would output the binary equivalent.

⇒ Program -

```
import java.util.ArrayDeque;
import java.util.Scanner;

public class DectoBin {
    public static void main(String args[]) {
        ArrayDeque stack = new ArrayDeque();
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter number: ");
        int number = Integer.valueOf(
                                scanner.nextLine());

        while (number > 0) {
            stack.addFirst (number % 2);
            number = number / 2;
        }
```

```
        System.out.println("Equivalent Binary: ");
        while(!stack.isEmpty()) {
            System.out.println(stack.removeFirst());
        }
    }
}
```

Q.6. Select and apply an appropriate data structures to store data in each of the following cases.

a) A list of employees records need to be stored in a manner that is easy to find max or min of the list.

Ans→ Since we need to find min and max of the list quickly we can use a specialized priority queue called Double Ended Priority Queue. We can implement it using LinkedList having pointers to both front and rear. Insertion will be O(n), but getMin() and getMax() will be O(1).

Here we have a class Employee having all applicable attributes and methods along with a compare method which determines the comparability of object and thus its priority.

```
class EmployeeList{
    DoubleEndedPriorityQueue<Employee> list;

    EmployeeList() {
        this.list = new DoubleEndedPriorityQueue<>();
    }

    void insertEmployee(Employee e) {
        list.insert(e);
    }

    Employee getMinEmployee() {
        return list.getMin();
    }

    Employee getMaxEmployee() {
        return list.getMax();
    }
}
```

b) A library needs to maintain books by their ISBN. Only thing important is finding them as soon as possible.

Ans → We can use a Map here which will map ISBNs to their respective books. Insertion and retrieval will both be O(1).

Here we have a Book class having all the applicable attributes and methods along with their ISBN numbers as attribute.

```
class Library {
    HashMap<String, String> bookMap;

    Library() {
        booksMap = new HashMap<>();
    }

    void insertBook(Book book) {
        booksMap.put(Book.isbn, book);
    }

    void getBook(String isbn) {
        booksMap.get(isbn);
    }
}
```

c) A data set needs to be maintained in order to find the median of the set quickly.

Ans → For quickly finding the median we can use Sorted Array. In this case the insertion will be O(logn) (using quicksort) and retrieval of median will be O(1).

```
class DataSet{
    ArrayList<Integer> data;

    DataSet() {
        data = new ArrayList<>();
    }

    void insertData(int d) {
        data.add(d);
        Collections.sort(data);
    }
```

```
int getMedian() {
    int size = data.size();
    if (size % 2 == 0) {
        return data.get(size / 2) +
               data.get((size/2) - 1);
    } else {
        return data.get(size/2);
    }
}
```