Date _____

OSA Assignment #1

Name : Hammad Hafeez
Reg No # FA22-BCS-186

Q1

(1) a-b/(c+d*e)

Input	Stack	Output
a		a
-	-	a
b	-	ab
/	-/	ab
(	-/(	ab
c	-/(	abc
+	-/(+	abc
d	-/(+	abcd
*	-/(+*	abcd
e	-/(+*	abcde
)	~~-/(+*)~~	abcde
	-/	abcde*+
		abcde*+/-

Final string : abcde*+/-

next page →

(2) ((a+b)*c-(d-e))/(f+g)

Input	Stack	Output
(	(	
(	((	
a	((	a
+	((+	a
b	((+	ab
)	((+)	ab+
*	(*	ab+
c	(*	ab+c
-	(-	ab+c*
(	(-(	ab+c*
d	(-(	ab+c*d
-	(-(-	ab+c*d
e	(-(-	ab+c*de
)	(-(-)	ab+c*de-
)	(-)	ab+c*de--
/	(/	ab+c*de--
f	(/	ab+c*de--f
+	(/(	ab+c*de--f
g	(/(+	ab+c*de--fg
)	(/(+)	ab+c*de--fg+
		ab+c*de--fg+/

Final String: ab+c*de--fg+/

Date

(3) ((((1+2)*3)+6)/(2+3)

Input	Stack	Output
(	(	
(	((	
(	(((	
1	((((	1
+	(((+	1
2	(((+	12
)	(((~~+~~)	12+
*	((*	12+
3	((*	12+3
)	(~~(*~~)	12+3*
+	(+	12+3*
6	(+	12+3*6
)	(~~+~~)	12+3*6+
/	/(	12+3*6+
(	/((	12+3*6+
2	/((	12+3*6+2
+	/(+	12+3*6+2
3	/(+	12+3*6+23
)	/(~~+~~)	12+3*6+23+
		12+3*6+23+/

Final String : 12+3*6+23+/

Q2

Infix to Prefix

```
Character Stack s;
String Infix = "      "   //enter your equation
Stringbuilder reverse = new Stringbuilder();
    reverse.append(Infix);reverse();
    String prefix;
for (char c : ccreverseExpression.length();c++)
    if(C is letter or digit)
        prefix += C
else if ( c is ')' )
    s.push (')');
else if (c is'(')
    while (!s isEmpty && !s.peek()==')')
        char temp = s.pop;
        prefix += temp;
    end while
end if
if (!S.isEmpty && !s.peek() == ')')
    print ("Brackets Invalid");

s.pop; // To remove ')' );
else
    while ( !s.isEmpty && Precedence(s.peek()) >=
                          Precedence(C))
        char temp = s.pop();
        prefix += temp;
end while
    s.push(c)
end while
end for
```

```
while (!s.isEmpty())
char temp = s.pop();
prefix += temp;
end while

Stringbuilder finalAns = new Stringbuilder();
finalAns.append(prefix).reverse();
```

Q3

Ans:

```
import java.util.Stack;

public class QueueUsingStacks{

    private Stack<Integer> stack1;
    private Stack<Integer> stack2;

    public QueueUsingStacks(){
        stack1 = new Stack<Integer>();
        stack2 = new Stack<Integer>();
    }

    public void enqueue(int number){

        while(!stack1.isEmpty()){
            stack2.push(stack1.pop());
        }
        stack2.push(number);

        while(!stack2.isEmpty()){
            stack1.push(stack2.pop());
        }
    }

    public int dequeue(){
        if(isEmpty()){
            throw new IllegalStateException();
        }
        return stack1.pop();
    }
}
```

```
public boolean isEmpty(){
    return stack1.isEmpty();
}

public int size(){
    return stack1.size();
}
```

Q4

Double ended queue is a type of queue in which insertion and removal of elements can either be performed from front or the rear. Thus, it does not follow which can access element only in the front using double ended queue you can access. It also has a dynamic size which makes it easier to implement in cases where size is yet to be known. Furthermore it has a constant time complexity of O(1). Some of the methods of a double ended queue are:

1. insertFront(item);
2. insertRear(item);
3. deleteFront();
4. deleteRear();
5. getFront();
6. getRear();
7. isEmpty();
8. size();

Deque (Double ended queue) can also be implemented using Arrays or Linked List.