**1)Trace the following code, showing the contents of the stack after each invocation:**
Stack stack = new Stack(); stack.push(new Character('A')); stack.push(new Character('B')); stack.push(new Character('C')); stack.pop(); stack.pop(); stack.push(new Character('D')); stack.push(new Character('E')); stack.push(new Character('F')); stack.pop(); stack.push(new Character('G')); stack.pop(); stack.pop(); stack.pop();\

1. After stack.push(new Character('A')): Stack:[A]
2. After stack.push(new Character('B')): Stack: [A, B]
3. After stack.push(new Character('C')): Stack: [A, B, C]
4. After stack.pop(): Stack: [A, B]
5. After another stack.pop(): Stack: [A]
6. After stack.push(new Character('D')): Stack: [A, D]
7. After stack.push(new Character('E')): Stack: [A, D, E]
8. After stack.push(new Character('F')): Stack: [A, D, E, F]
9. After stack.pop(): Stack: [A, D, E]
10. After stack.push(new Character('G')): Stack: [A, D, E, G]
11. After another stack.pop(): Stack: [A, D, E]
12. After another stack.pop(): Stack: [A, D]
13. After the final stack.pop(): Stack: [A]

**2- Suppose an initially empty ArrayStack S has performed a total of 25 push operations, 12 top operations, and 10 pop operations, 3 of which returned null to indicate an empty stack. What is the current size of S? And what is the value of the instance variable t?**

Current size of S = 25 - 10 Current size of S = 15 Since there have been 15 elements pushed onto the stack and 7 elements popped, the index of the top element would be 15 - 7 - 1 (subtracting 1 since the index is zero-based). Value of t = 15 - 7 - 1 Value of t = 7

**3- Evaluate the following postfix expressions (true or false):**
a. 8 2 + 3 * 16 4 / - =26
b. 12 2 5 5 1 / / * 8 7 + - =- 9
c. 70 14 4 5 15 3 / * - / 6 + =(لا نستطيع تقسيم عدد على الصفر) ∞
d. 3 5 6 * + 13 - 18 2 / + =29

**4)Convert the following infix expressions to postfix notations, and convert the first**
a. (A + B) * (C + D) - E
b. A - (B + C) * D + E / F
c. ((A + B) / (C - D) + E) * F - G
d. A + B * (C + D) - E / F * G + H

```
import java.util.Deque;
import java.util.LinkedList;
Stack<Character> stack = new Stack<>();
        StringBuilder postfix = new StringBuilder();
        String expression = "(A + B) * (C + D) - E";
        for (char c : expression.toCharArray()) {
        if (Character.isLetterOrDigit(c)) {
        postfix.append(c);
        } else if (c == '(') {
        stack.push(c);
        } else if (c == ')') {
        while (!stack.isEmpty() && stack.peek() != '(') {
        postfix.append(stack.pop());
```

```
}
stack.pop();
} else {
while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
postfix.append(stack.pop());
}
stack.push(c);
}
}
while (!stack.isEmpty()) {
postfix.append(stack.pop());
}
String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```
Note: The `precedence()` method is used to determine the precedence of operators.
b. A - (B + C) * D + E / F
Postfix notation: ABC+D*-EF/+
Java code using stack operations:
````java
Stack<Character> stack = new Stack<>();
StringBuilder postfix = new StringBuilder();
String expression = "A - (B + C) * D + E / F";
for (char c : expression.toCharArray()) {
if (Character.isLetterOrDigit(c)) {
postfix.append(c);
} else if (c == '(') {
stack.push(c);
} else if (c == ')') {
while (!stack.isEmpty() && stack.peek() != '(') {
postfix.append(stack.pop());
}
stack.pop();
} else {
while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
postfix.append(stack.pop());
}
stack.push(c);
}
}
while (!stack.isEmpty()) {
postfix.append(stack.pop());
}
String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```
c. ((A + B) / (C - D) + E) * F - G
Postfix notation: AB+CD-/E+F*GJava code using stack operations:
````java
Stack<Character> stack = new Stack<>();
StringBuilder postfix = new StringBuilder();
String expression = "((A + B) / (C - D) + E) * F - G";
for (char c : expression.toCharArray()) {
if (Character.isLetterOrDigit(c)) {
postfix.append(c);
} else if (c == '(') {
```

```
stack.push(c);
} else if (c == ')') {
while (!stack.isEmpty() && stack.peek() != '(') {
postfix.append(stack.pop());
}
stack.pop();
} else {
while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
postfix.append(stack.pop());
}
stack.push(c);
}
}
while (!stack.isEmpty()) {
postfix.append(stack.pop());
}
String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```
d. A + B * (C + D) - E / F * G + H
Postfix notation: ABCD+*+EF/G*-H+
Java code using stack operations:
````java
Stack<Character> stack = new Stack<>();
StringBuilder postfix = new StringBuilder();
String expression = "A + B * (C + D) - E / F * G + H";
for (char c : expression.toCharArray()) {
if (Character.isLetterOrDigit(c)) {
postfix.append(c);
} else if (c == '(') {
stack.push(c);
} else if (c == ')') {
while (!stack.isEmpty() && stack.peek() != '(') {
postfix.append(stack.pop());
}
stack.pop();
} else {
while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {
postfix.append(stack.pop());
}
stack.push(c);
}
}
while (!stack.isEmpty()) {
postfix.append(stack.pop());
}
String postfixExpression = postfix.toString();
System.out.println(postfixExpression);
```

**5- Write the definition of the function template printListReverse that uses a stack to print a linked list in reverse order. Assume that this function is a member of the class linkedStack,**

```
template <class T>
class linkedStack {
   private:
   struct Node {
      T data;
```

```
    Node* next;
};
Node* top;
public:
// Other member functions of linkedStack
void printListReverse() {
    std::stack<T> stack;
    Node* temp = top;
    // Push elements of linked list onto the stack
    while (temp != nullptr) {
        stack.push(temp->data);
        temp = temp->next;
    }
```