**1\**

1. ()
2. (A)
3. ((A,B,C
4. A,B))
5. (A)
6. A,D))
7. (A,D,E)
8. (A,D,E,F)
9. (A,D,E)
10. (A,D,E,G)
11. (A,D,E)
12. A,D))
13. (A)

**2\**

Size=18 , t=17

**3\**

1. T
2. F
3. F
4. T

**4\**

a. (A + B) \* (C + D) - E

التعبير المرجعي المُحول: AB+CD+\*E-

b. A - (B + C) \* D + E / F

التعبير المرجعي المُحول: ABC+D\*-EF/+

c. ((A + B) / (C - D) + E) \* F - G

التعبير المرجعي المُحول: AB+CD-/E+F\*G-

d. A + B \* (C + D) - E / F \* G + H

التعبير المرجعي المُحول: ABCD+\*+EF/G\*-H+

والآن سأقوم بتحويل التعابير المرجعية الأوليتين إلى كود جافا باستخدام عمليات الستاك:

1. التعبير المرجعي المُحول: AB+CD+\*E-

**```java**

**import java.util.Stack;**

**public class Main {**

**public static void main(String[] args) {**

**String expression = "AB+CD+\*E-";**

**Stack<Character> stack = new Stack<>();**

**for (char c : expression.toCharArray()) {**

**if (Character.isLetter(c)) {**

**stack.push(c);**

**} else {**

**char operand2 = stack.pop();**

**char operand1 = stack.pop();**

**char result = performOperation(operand1, operand2, c);**

**stack.push(result);**

**}**

**}**

**char finalResult = stack.pop();**

**System.out.println("Final result: " + finalResult);**

**}**

**private static char performOperation(char operand1, char operand2, char operator) {**

**// Perform the operation based on the operator**

**// You can define your own logic here**

**return 'X'; // Replace 'X' with the actual result**

**}**

**}**

**```**

2. التعبير المرجعي المُحول: ABC+D\*-EF/+

**```java**

**import java.util.Stack;**

**public class Main {**

**public static void main(String[] args) {**

**String expression = "ABC+D\*-EF/+";**

**Stack<Character> stack = new Stack<>();**

**for (char c : expression.toCharArray()) {**

**if (Character.isLetter(c)) {**

**stack.push(c);**

**} else {**

**char operand2 = stack.pop();**

**char operand1 = stack.pop();**

**char result = performOperation(operand1, operand2, c);**

**stack.push(result);**

**}**

**}**

**char finalResult = stack.pop();**

**System.out.println("Final result: " + finalResult);**

**}**

**private static char performOperation(char operand1, char operand2, char operator) {**

**// Perform the operation based on the operator**

**// You can define your own logic here**

**return 'X'; // Replace 'X' with the actual result**

**} }**

**5\**

**#include <iostream>**

**#include <stack>**

**template <class T>**

**class linkedStack {**

**private:**

**struct Node {**

**T data;**

**Node\* next;**

**};**

**Node\* top;**

**public:**

**// Constructor**

**linkedStack() {**

**top = nullptr;**

**}**

**// Function to print the linked list in reverse order**

**void printListReverse() {**

**std::stack<T> stack;**

**// Push each element of the linked list onto the stack**

**Node\* current = top;**

**while (current != nullptr) {**

**stack.push(current->data);**

**current = current->next;**

**}**

**// Pop and print each element from the stack**

**while (!stack.empty()) {**

**std::cout << stack.top() << " ";**

**stack.pop();**

**}**

**}**

**// Other member functions of linkedStack...**

**};**

**6\**

**import java.util.Stack;**

**public class Main {**

**public static <E> void reverse(Stack<E> stack) {**

**Stack<E> tempStack = new Stack<>();**

**// Push each element from the original stack to the temporary stack**

**while (!stack.isEmpty()) {**

**tempStack.push(stack.pop());**

**}**

**// Pop and push each element from the temporary stack back to the original stack**

**while (!tempStack.isEmpty()) {**

**stack.push(tempStack.pop());**

**}**

**}**

**public static void main(String[] args) {**

**Stack<Integer> stack = new Stack<>();**

**stack.push(1);**

**stack.push(2);**

**stack.push(3);**

**stack.push(4);**

**System.out.println("Original Stack: " + stack);**

**reverse(stack);**

**System.out.println("Reversed Stack: " + stack);**

**}**

**}**

**7\**

**import java.util.Stack;**

**public class Main {**

**public static <E> E popBottom(Stack<E> stack) {**

**if (stack.isEmpty()) {**

**throw new IllegalStateException("Stack is empty");**

**}**

**if (stack.size() == 1) {**

**return stack.pop();**

**}**

**E topElement = stack.pop();**

**E bottomElement = popBottom(stack);**

**stack.push(topElement);**

**return bottomElement;**

**}**

**public static void main(String[] args) {**

**Stack<Integer> stack = new Stack<>();**

**stack.push(1);**

**stack.push(2);**

**stack.push(3);**

**stack.push(4);**

**System.out.println("Original Stack: " + stack);**

**Integer bottomElement = popBottom(stack);**

**System.out.println("Bottom Element: " + bottomElement);**

**System.out.println("Updated Stack: " + stack);**

**}**

**}**

**8\**

**import java.util.EmptyStackException;**

**public class ArrayStack<E> {**

**private E[] elements;**

**private int top;**

**public ArrayStack(int capacity) {**

**elements = (E[]) new Object[capacity];**

**top = -1;**

**}**

**public boolean isEmpty() {**

**return top == -1;**

**}**

**public boolean isFull() {**

**return top == elements.length - 1;**

**}**

**public void push(E item) {**

**if (isFull()) {**

**throw new IllegalStateException("Stack is full");**

**}**

**elements[++top] = item;**

**}**

**public E pop() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return elements[top--];**

**}**

**public E top() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return elements[top];**

**}**

**public E topSecond() {**

**if (top < 1) {**

**throw new IllegalStateException("Stack does not have a second element");**

**}**

**return elements[top - 1];**

**}**

**}**

**9\**

**import java.util.EmptyStackException;**

**public class ArrayStack<E> {**

**private E[] elements;**

**private int top;**

**public ArrayStack(int capacity) {**

**elements = (E[]) new Object[capacity];**

**top = -1;**

**}**

**public boolean isEmpty() {**

**return top == -1;**

**}**

**public boolean isFull() {**

**return top == elements.length - 1;**

**}**

**public void push(E item) {**

**if (isFull()) {**

**throw new IllegalStateException("Stack is full");**

**}**

**elements[++top] = item;**

**}**

**public E pop() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return elements[top--];**

**}**

**public E top() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return elements[top];**

**}**

**public E topSecond() {**

**if (top < 1) {**

**throw new IllegalStateException("Stack does not have a second element");**

**}**

**return elements[top - 1];**

**}**

**public E popSecond() {**

**if (top < 1) {**

**throw new IllegalStateException("Stack does not have a second element");**

**}**

**E secondElement = elements[top - 1];**

**for (int i = top - 1; i < top; i++) {**

**elements[i] = elements[i + 1];**

**}**

**top--;**

**return secondElement;**

**}**

**}**

**10\**

**import java.util.EmptyStackException;**

**public class LinkedStack<E> {**

**private Node<E> top;**

**private int size;**

**private static class Node<E> {**

**E data;**

**Node<E> next;**

**public Node(E data, Node<E> next) {**

**this.data = data;**

**this.next = next;**

**}**

**}**

**public LinkedStack() {**

**top = null;**

**size = 0;**

**}**

**public boolean isEmpty() {**

**return size == 0;**

**}**

**public void push(E item) {**

**Node<E> newNode = new Node<>(item, top);**

**top = newNode;**

**size++;**

**}**

**public E pop() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**E poppedData = top.data;**

**top = top.next;**

**size--;**

**return poppedData;**

**}**

**public E top() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return top.data;**

**}**

**public E bottom() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**Node<E> currentNode = top;**

**while (currentNode.next != null) {**

**currentNode = currentNode.next;**

**}**

**return currentNode.data;**

**}**

**}**

**11\**

**import java.util.EmptyStackException;**

**public class ArrayStack<E> {**

**private E[] elements;**

**private int top;**

**public ArrayStack(int capacity) {**

**elements = (E[]) new Object[capacity];**

**top = -1;**

**}**

**public boolean isEmpty() {**

**return top == -1;**

**}**

**public boolean isFull() {**

**return top == elements.length - 1;**

**}**

**public void push(E item) {**

**if (isFull()) {**

**throw new IllegalStateException("Stack is full");**

**}**

**elements[++top] = item;**

**}**

**public E pop() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return elements[top--];**

**}**

**public E top() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return elements[top];**

**}**

**public E bottom() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**return elements[0];**

**}**

**public E popbottom() {**

**if (isEmpty()) {**

**throw new EmptyStackException();**

**}**

**E bottomElement = elements[0];**

**for (int i = 0; i < top; i++) {**

**elements[i] = elements[i + 1];**

**}**

**top--;**

**return bottomElement;**

**}**

**}**

**12\**

**a) محتويات الستاك:**

**محتويات الستاك ستكون: [2, 4, 8, 10, 14, 16]**

**b) قيم المتغيرات:**

**القيمة النهائية للمتغير `count` ستكون 6.**

**القيمة النهائية للمتغير `top` ستكون 5.**

**c) عنصر الدالة `top()` في الستاك:**

**عنصر الدالة `top()` في الستاك سيكون 16.**

**d) هل الستاك ممتلئ؟ ولماذا؟**

**الستاك ليس ممتلئًا. الستاك لديه قدرة بحجم 10 ولديه حاليًا 6 عناصر فقط.**

**e) جعل الستاك يعود إلى الحالة الفارغة:**

**لجعل الستاك يعود إلى الحالة الفارغة، يمكن استدعاء دالة `pop()` حتى يتم إزالة جميع العناصر من الستاك. يمكن استخدام حلقة تكرارية للقيام بذلك، كالتالي:**

**```java**

**while (!stack.isEmpty()) {**

**stack.pop();**

**}**

**```**

**بعد تنفيذ هذا الكود، ستاك ستعود إلى الحالة الفارغة.**