**Answer Question:1**

1. Initial state: Empty stack

2. stack.push(new Character('A')):

Stack contents: A

3. stack.push(new Character('B')):

Stack contents: A, B

4. stack.push(new Character('C')):

Stack contents: A, B, C

5. stack.pop():

Stack contents: A, B

6. stack.pop():

Stack contents: A

7. stack.push(new Character('D')):

Stack contents: A, D

8. stack.push(new Character('E')):

Stack contents: A, D, E

9. stack.push(new Character('F')):

Stack contents: A, D, E, F

10. stack.pop():

Stack contents: A, D, E

11. stack.push(new Character('G')):

Stack contents: A, D, E, G

12. stack.pop():

Stack contents: A, D, E

13. stack.pop():

Stack contents: A, D

14. stack.pop():

Stack contents: A

15. Empty stack

**Question:2**

Current size of S = 25 - 10 Current size of S = 15

Value of t = 7

**Question:3**

**a/true(26)**

**, b/ true(35)**

**, c/true(-4)**

**, d/true(29)**

**Question:5**

**template <typename T>**

**void linkedStack<T>::printListReverse() const {**

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

**Node<T>\* current = top; // Assuming top is the head of the linked list**

**// Push elements of the linked list onto the stack**

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

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

**current = current->next;**

**}**

**// Pop elements from the stack and print them in reverse order**

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

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

**stack.pop();**

**}**

**std::cout << std::endl;**

**}**

**Question:6**

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

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

**return; // Nothing to reverse if the stack is empty }**

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

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

**E element = stack.top();**

**tempStack.push(element);**

**stack.pop();}**

**// Move the elements from the temporary stack back to the original stack**

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

**E element = tempStack.top();**

**stack.push(element);**

**tempStack.pop(); }}**

**Question:7**

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

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

**throw new NoSuchElementException("Stack is empty");}**

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

**// Transfer elements from the original stack to the temporary stack**

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

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

**stack.pop();}**

**// Remove and return the bottom element from the temporary stack**

**E bottomElement = tempStack.pop();**

**// Transfer elements back from the temporary stack to the original stack**

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

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

**tempStack.pop();}**

**return bottomElement; }**

**Question:8**

**public class ArrayStack<E> {**

**private E[] elements;**

**private int top;**

**// constructor and other methods**

**public E topSecond() {**

**if (top < 1) {**

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

**}**

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

**}**

**}**

**Question:9**

**public class ArrayStack<E> {**

**private E[] elements;**

**private int top;**

**// constructor and other methods**

**public E popSecond() {**

**if (top < 1) {**

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

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

**top--;**

**return secondElement; }}**

**Question:10**

**public class LinkedStack<E> {**

**private Node<E> top;**

**// constructor and other methods**

**public E bottom() {**

**if (top == null) {**

**throw new NoSuchElementException("Stack is empty"); }**

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

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

**current = current.next; }**

**return current.data; }**

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

**private E data;**

**private Node<E> next;**

**// constructor and other methods }}**

**Question:11**

**public class ArrayStack<E> {**

**private E[] elements;**

**private int top;**

**// constructor and other methods**

**public E popbottom() {**

**if (top < 1) {**

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

**}**

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

**// Shift all elements to the left to remove the bottom element**

**for (int i = 0; i < top - 1; i++) {**

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

**}**

**top--; // Decrement the top index**

**return bottomElement;**

**}**

**}**