**Batch: C3 Roll No.:16010123217**

**Experiment / assignment / tutorial No. 4**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Dynamic implementation of Stack- Creation, Insertion, Deletion, Peek |

**Objective:** To implement Basic Operations of Stack dynamically

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 2 | Apply linear and non-linear data structure in application development. |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. [*https://www.cprogramming.com/tutorial/computersciencetheory/stack.html*](https://www.cprogramming.com/tutorial/computersciencetheory/stack.html)
3. [*https://www.geeksforgeeks.org/stack-data-structure-introduction-program/*](https://www.geeksforgeeks.org/stack-data-structure-introduction-program/)
4. [*https://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html*](https://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html)
5. [*https://www.naukri.com/code360/library/application-of-stack*](https://www.naukri.com/code360/library/application-of-stack)
6. [*https://www.enjoyalgorithms.com/blog/application-of-stack-data-structure-in-programming*](https://www.enjoyalgorithms.com/blog/application-of-stack-data-structure-in-programming)
7. [*https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-using-stack/*](https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-using-stack/)

**Abstract**:

A Stack is an ordered collection of elements , but it has a special feature that deletion and insertion of elements can be done only from one end, called the top of the stack(TOP). The order may be LIFO(Last In First Out) or FILO(First In Last Out).

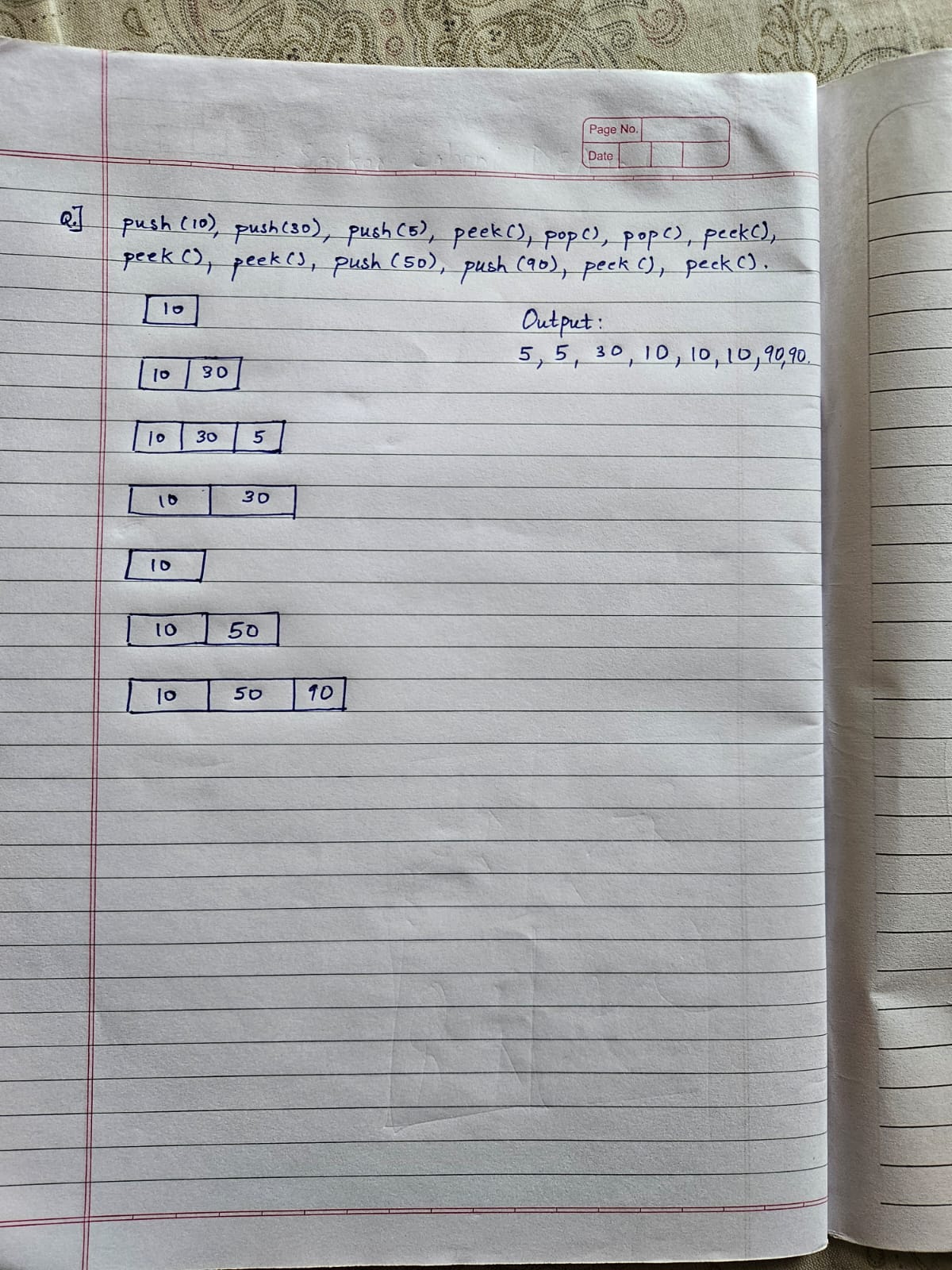
**Related Theory: -**

Stack is a linear data structure which follows a particular order in which the operations are performed. It works on the mechanism of Last in First out (LIFO).

**List 5 Real Life applications of stack:**

1. Back and Forward buttons in a web browser
2. UNDO/REDO functionality in text editors and image editing software
3. Memory management in computer programming
4. Implementing recursion in programming
5. Matching HTML tags in web development

**List and perform some push and pop operations on a stack that undergoes dynamic memory allocation: (** *Students may solve this with pen-paper and paste a pic of the work here)*



**Algorithm for creation, insertion, deletion, displaying an element in dynamic stack:**

**Creation (CreateStack):**

* Input: A pointer to the Stack s.
* Operation: Set s->top to NULL.
* Output: The stack is initialized and empty.

**Insertion (Push):**

* Input: A pointer to the Stack s and an integer ele to push.
* Operation:
  1. Allocate memory for a new node.
  2. If memory allocation fails, print an error message and exit the operation.
  3. Set the data of the new node to ele.
  4. Set the next of the new node to point to s->top.
  5. Update s->top to point to the new node.
* Output: The element ele is pushed onto the stack.

**Deletion (Pop):**

* **Input:** A pointer to the Stack s.
* **Operation:**
  1. If the stack is empty, print "Stack Underflow" and return -1.
  2. Store the data of the top node in a variable poppedData.
  3. Update s->top to point to the next node.
  4. Free the memory of the original top node.
  5. Return the poppedData.
* **Output:** The top element is removed from the stack and returned. If the stack is empty, return -1.

**Displaying an Element (Peek):**

* **Input:** A pointer to the Stack s.
* **Operation:**
  1. If the stack is empty, print "Stack is empty" and return -1.
  2. Return the data of the top node.
* **Output:** The top element of the stack is returned. If the stack is empty, return -1.

**Program source code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

typedef struct {

Node \*top;

} Stack;

void initialize(Stack \*s) {

s->top = NULL;

}

int isEmpty(Stack \*s) {

return s->top == NULL;

}

void push(Stack \*s, int ele) {

Node \*newNode = (Node \*)malloc(sizeof(Node));

if (!newNode) {

printf("Memory allocation failed\n");

return;

}

newNode->data = ele;

newNode->next = s->top;

s->top = newNode;

printf("Pushed %d to the stack\n", ele);

}

int pop(Stack \*s) {

if (isEmpty(s)) {

printf("Stack Underflow\n");

return -1;

}

Node \*temp = s->top;

int poppedData = temp->data;

s->top = s->top->next;

free(temp);

return poppedData;

}

int peek(Stack \*s) {

if (isEmpty(s)) {

printf("Stack is empty\n");

return -1;

}

return s->top->data;

}

void delete(Stack \*s) {

while (!isEmpty(s)) {

pop(s);

}

printf("Stack deleted\n");

}

void display(Stack \*s) {

if (isEmpty(s)) {

printf("Stack is empty\n");

return;

}

Node \*temp = s->top;

printf("Stack elements: ");

while (temp) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

int main() {

Stack s;

initialize(&s);

int choice, ele;

while (1) {

printf("\nMenu:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. Delete stack\n");

printf("5. Display stack\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the element to be pushed: ");

scanf("%d", &ele);

push(&s, ele);

break;

case 2:

ele = pop(&s);

if (ele != -1) {

printf("Popped element: %d\n", ele);

}

break;

case 3:

ele = peek(&s);

if (ele != -1) {

printf("Top element: %d\n", ele);

}

break;

case 4:

delete(&s);

break;

case 5:

display(&s);

break;

case 6:

delete(&s);

exit(0);

default:

printf("Invalid choice, please try again\n");

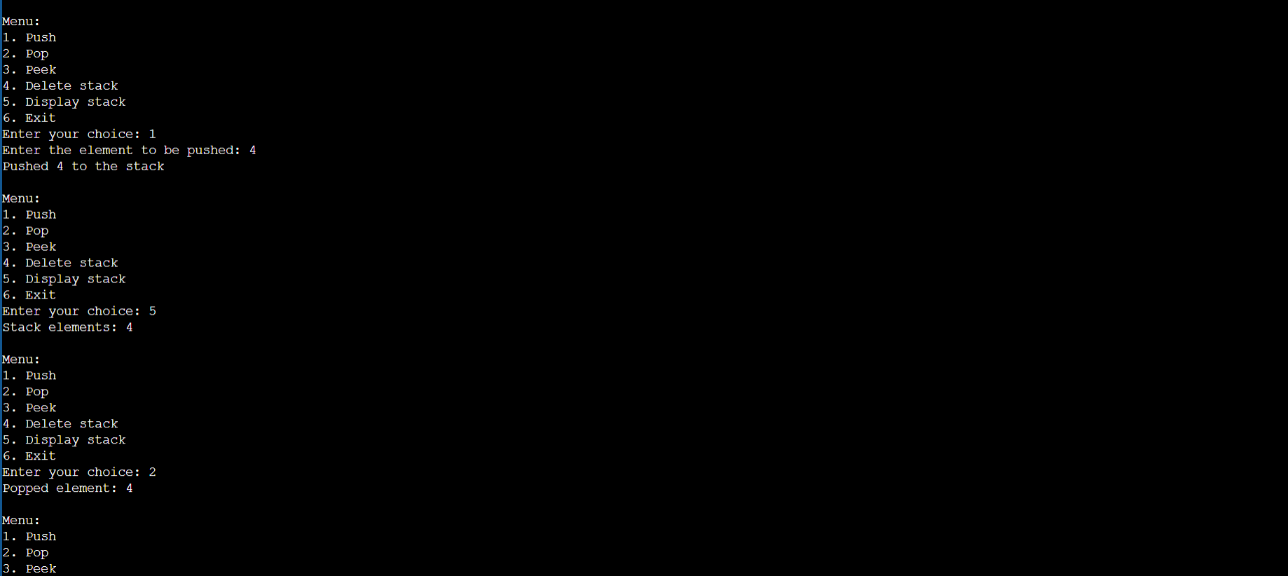
}

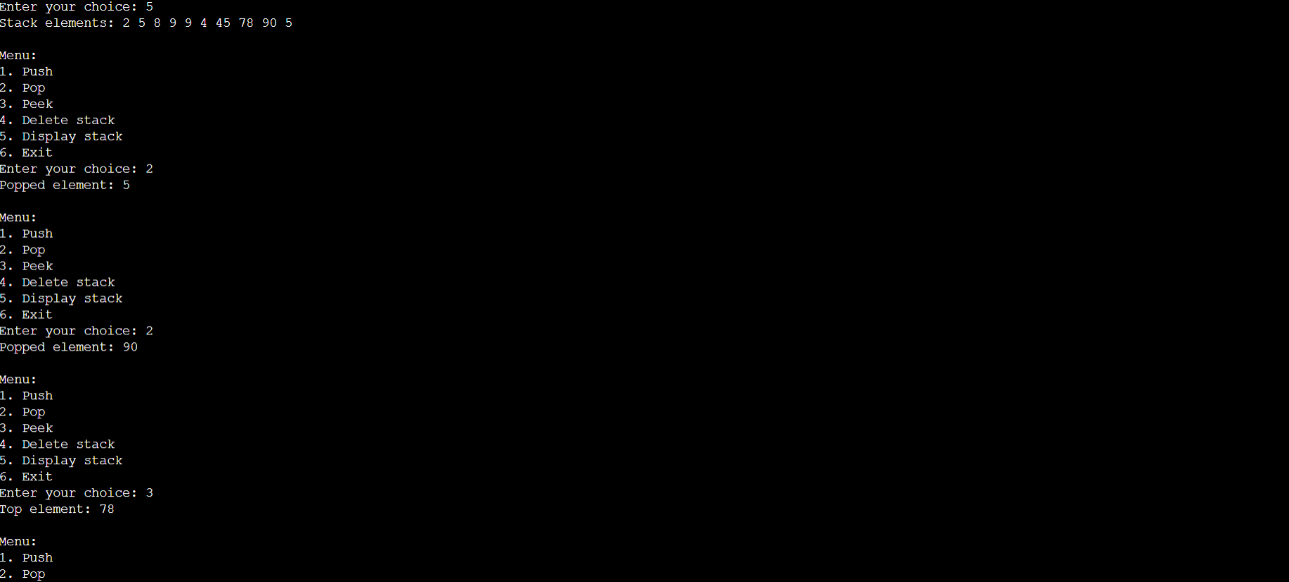
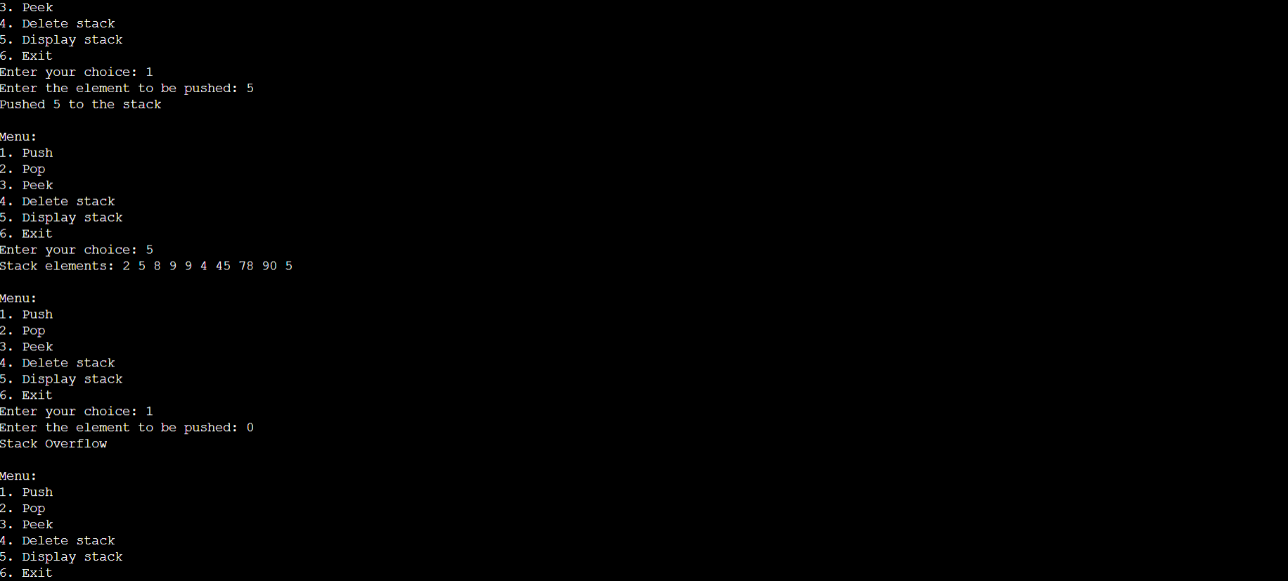
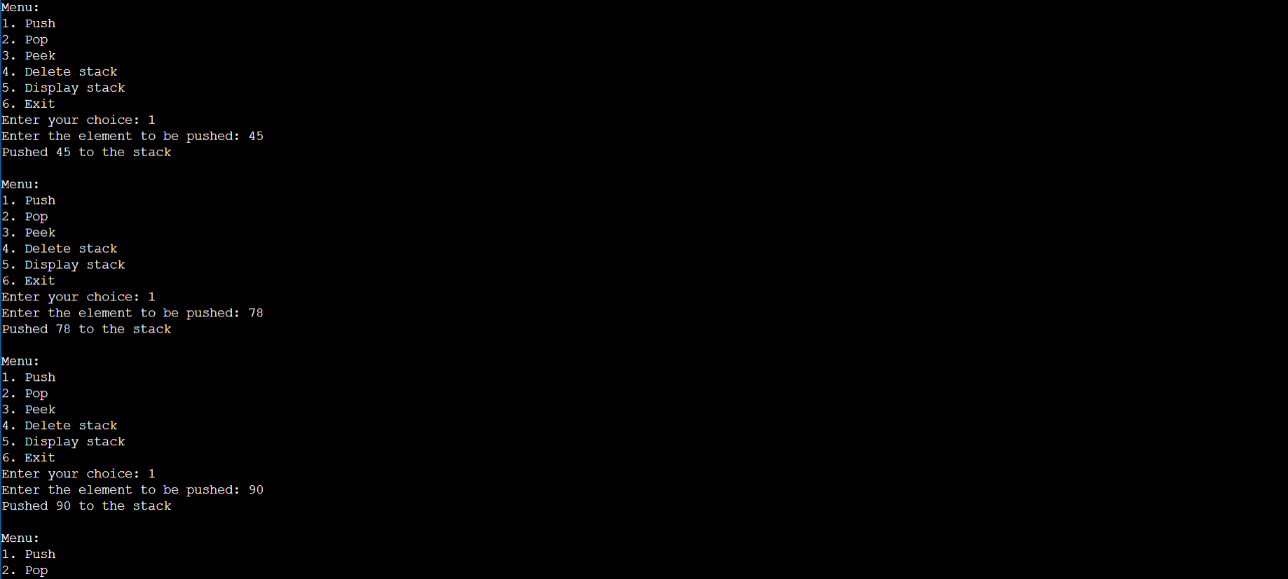
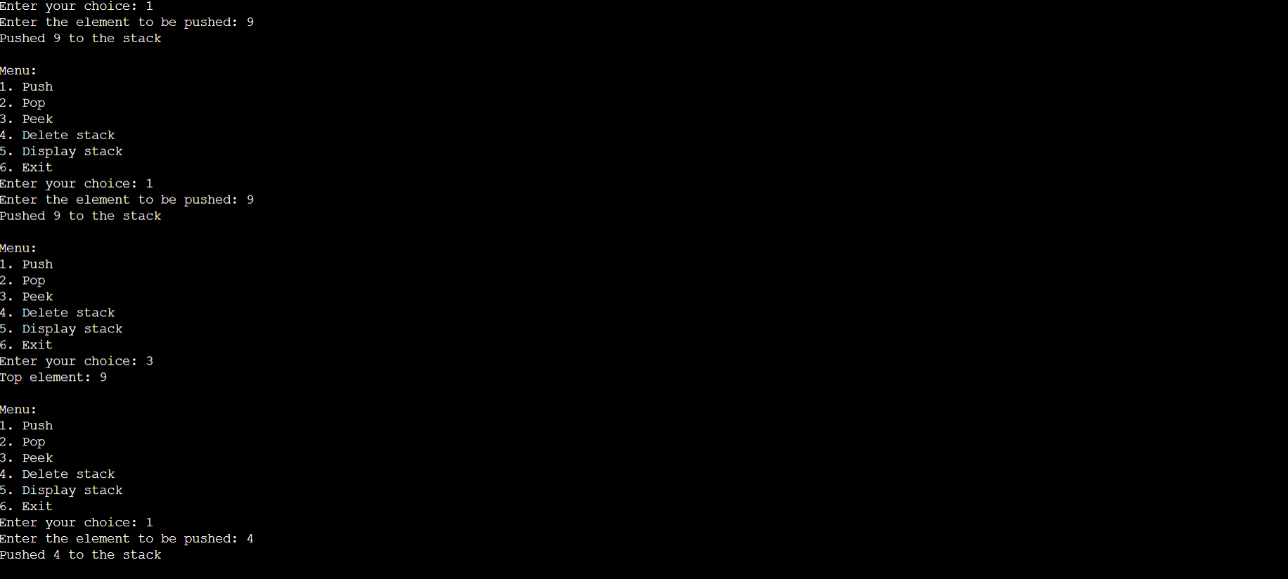
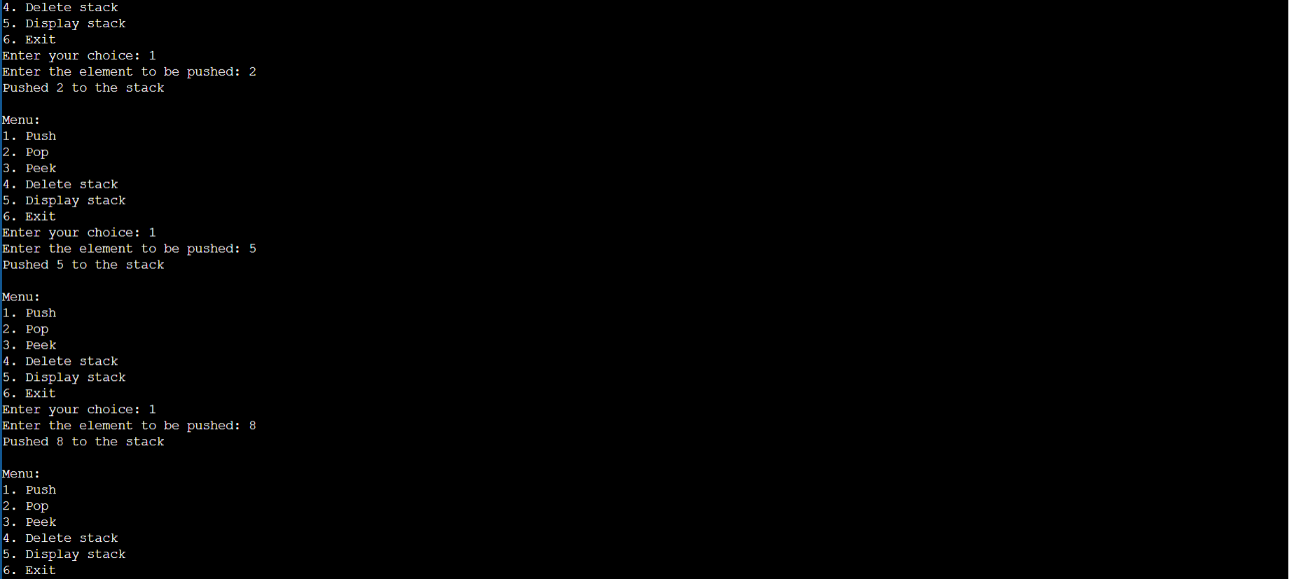
}

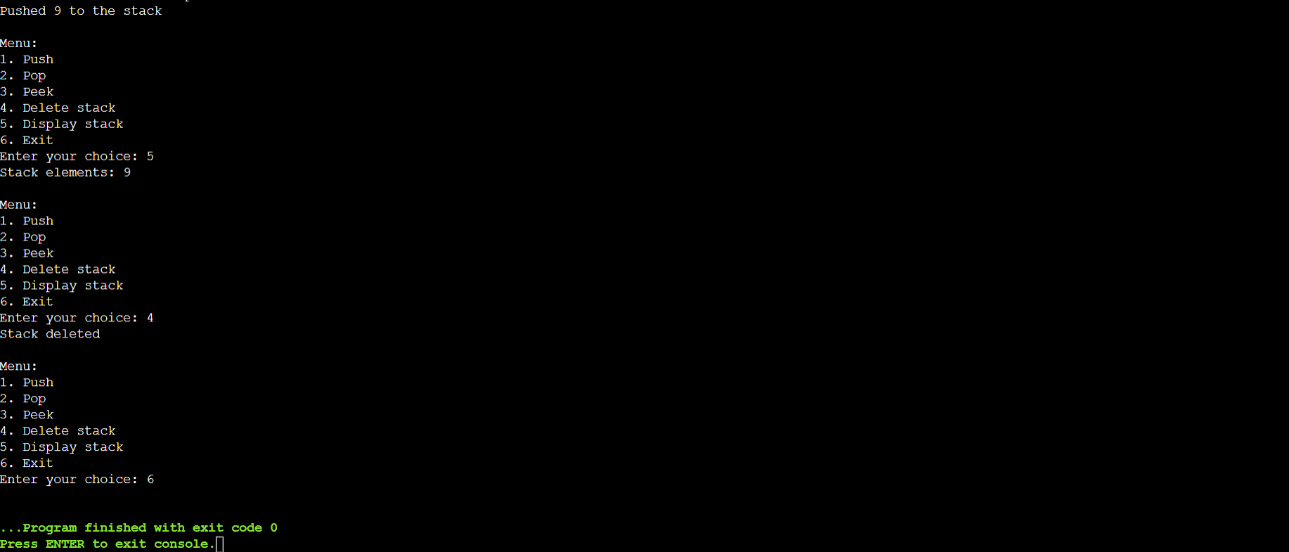
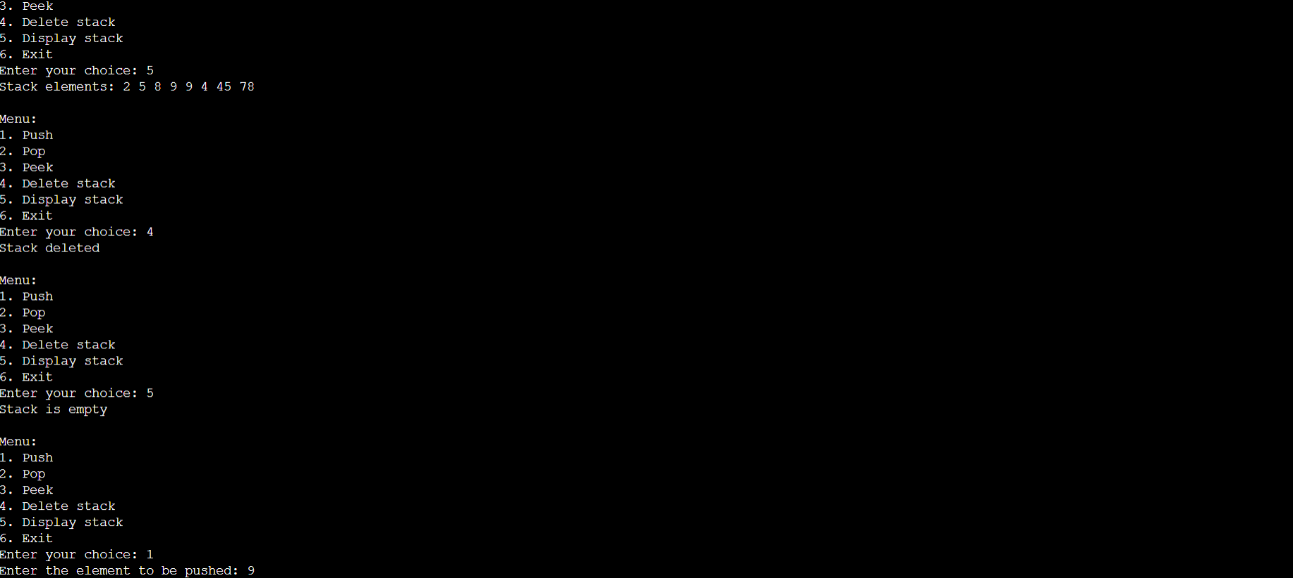
return 0;

}

**Output Screenshots:**

****

****

****

**Conclusion:-**

**In this experiment how to build stack data structure using dynamic memory allocation**

**Post Lab Questions:**

1. **Explain how Stacks can be used in Backtracking algorithms with example.**

**Ans.** Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

Uses stack for storing solution path

Source: <https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-using-stack/>

* Using a Stack in Backtracking:

In the "Rat in a Maze" problem, a rat needs to find a path from the top-left corner to the bottom-right corner of a grid, avoiding obstacles. The stack is used to store the positions the rat has visited, helping the algorithm to backtrack when it hits a dead-end.

Algorithm Outline:

Start from the initial position (0,0).

Push the current position onto the stack.

Check all possible moves (down, right, up, left) from the current position.

If a move is valid and leads to an open path, push the new position onto the stack.

If no move is possible, pop the stack to backtrack to the previous position.

Repeat until the rat reaches the destination or all paths are exhausted.

Example:

Consider a 4x4 maze:

1 0 0 0

1 1 0 1

0 1 0 0

1 1 1 1

1 represents a path the rat can take.

0 represents an obstacle.

Steps Using Stack:

Start at (0,0): Push (0,0) onto the stack.

Move to (1,0): Push (1,0).

Move to (1,1): Push (1,1).

Move to (2,1): Push (2,1).

Move to (3,1): Push (3,1).

Move to (3,2): Push (3,2).

Move to (3,3): Push (3,3). (Destination reached)

If the rat hits a dead-end, such as trying to move from (2,1) to (2,2) and finding an obstacle, it would pop (2,2) and try the next possible move, using the stack to track and reverse its steps.

1. **Implement any one of the stack application, paste code and output screenshots.**

**Ans:** Multiple parenthesis check

Using inbuilt stack data structure in c++

Code   
#include <bits/stdc++.h>

using namespace std;

bool check(string s){

if(s.length()%2!=0){

return false;

}

stack<char>a;

for(int i=0;i<s.length()/2;i++){

a.push(s[i]);

}

for(int i=s.length()/2;i<s.length();i++){

if((a.top()=='[' and s[i]==']') || (a.top()=='{' and s[i]=='}') || (a.top()=='(' and s[i]==')')){

a.pop();

}

}

if(a.empty()){

return true;

}

else{

return false;

}

}

int main()

{

string s="{[()]}";

if(check(s)){

cout<<"The given expression "<<s<<" is balanced";

}else{

cout<<"The given expression"<<s<<" is not balanced";

}

return 0;

}

Output  
