K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: A2 Roll No.: 16010123032
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
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.cprogramming.com/tutorial/computersciencetheory/stack.html
5. https://www.geeksforgeeks.org/stack-data-structure-introduction-program/
6. https://www.thecrazyprogrammer.com/2013/12/c-program-for-array-representation-of-stack-push-pop-display.html

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).
Students need to first try and understand the implementation of using arrays. Once comfortable with the concept, they can further implement stacks using linked list as well.

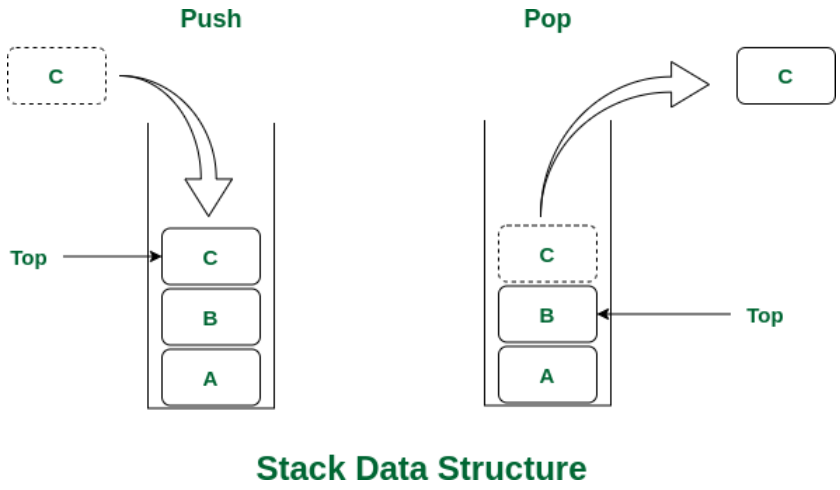
Related Theory: -

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 Examples:

1. Undo/Redo Feature in all Softwares.
2. For managing history in web browsers (back and forward button)
3. OS uses stacks to manage processes.
4. IDE's use stack to highlight syntax and match opening and closing elements.
5. It is used in various algorithms like DFS (Depth First Search) and Backtracking.

Diagram:



Stack Data Structure

Explain Stack ADT:

Stack Data Structure is an Abstract Data Type that follows LIFO (Last-in First-Out) Principle. It is a linear data structure. In this data type, only the top element can be accessed, peeked, removed or added.

Some of the key operations are:

1. Pop
2. Peek
3. Push
4. Display Elements

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

Node

data: Integer data to be stored in the node.

next: Pointer to the next node in the stack.

**top: Pointer to the top node of the stack.

Push

1. Create a new node.
2. Assign the data to the new node.
3. Make the new node point to the current top.
4. Update the top to point to the new node.

Pop

1. Check if the stack is empty (top is NULL).

   If empty, return an error (e.g., -1).
2. Store the data of the top node.
3. Make the top point to the next node.
4. Free the memory of the popped node.
5. Return the stored data.

Display

1. Initialize a temporary pointer to the top.
2. While the temporary pointer is not NULL:

   Print the data of the current node.

   Move the temporary pointer to the next node.

Peek

1. Check if the stack is empty (top is NULL).

   If empty, print an error message.
2. Print the data of the top node.

Program source code:

```
#include <stdio.h>
#include <stdlib.h>

void push();
int pop();
void display();
void peek();

struct NODE
{
   int data;
   struct NODE *next;
} *top = NULL, *ptr, *temp;
```

```
void push()
{
   int ele;
   printf("\nEnter the element to be pushed: ");
   scanf("%d", &ele);

   ptr = (struct NODE *)malloc(sizeof(struct NODE));
   ptr->data = ele;
   ptr->next = top;
   top = ptr;
}

int pop()
{
   int val;
   if (top == NULL) {
      printf("\nStack underflow");
      return -1; // Returning -1 to indicate an error, handle accordingly
   }
   else
   {
      temp = top;
      val = temp->data;
      top = top->next;
      free(temp);
      return val;
```

```
    }
}

void display()
{
    temp = top;
    if (temp == NULL)
    {
        printf("\nStack is empty");
    }
    else
    {
        while (temp != NULL)
        {
            printf("\n%d", temp->data);
            temp = temp->next;
        }
    }
}

void peek()
{
    if (top == NULL)
    {
        printf("\nStack is empty");
    } else
    {
```

```
        printf("\nTop element is %d", top->data);
    }
}

int main()
{
    int ch, i;
    while (1)
    {
        printf("\nMENU OPTION\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Peek\n");
        printf("5. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &ch);
        if (ch < 1 || ch > 5)
        {
            printf("\nInvalid choice\n");
            continue;
        }
        switch (ch)
        {
            case 1:
                push();
```

```
            break;
        case 2:
            i = pop();
            if (i != -1)
            {
                printf("%d item deleted\n", i);
            }
            printf("After removing\n");
            display();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            exit(0);
      }
   }
}
```

Output Screenshots:

```
MENU OPTION
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1

Enter the element to be pushed: 12

MENU OPTION
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1

Enter the element to be pushed: 3

MENU OPTION
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 1

Enter the element to be pushed: 7

MENU OPTION
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 2
7 item deleted
After removing

3
12
MENU OPTION
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 4

Top element is 3
MENU OPTION
1. Push
2. Pop
3. Display
4. Peek
5. Exit
Enter your choice: 5


=== Code Execution Successful ===
```

Conclusion:-
Learnt to implement Basic Operations of Stack dynamically**.**

PostLab Questions:

1) **Explain how Stacks can be used in Backtracking algorithms with example.**
2) **Illustrate the concept of Call stack in Recursion.**

Answers:

1)

Backtracking is a method for solving problems incrementally by trying partial solutions and abandoning them if they don't lead to a valid solution. A **stack** is a data structure that follows Last In, First Out (LIFO) order and is particularly useful in backtracking algorithms to manage state exploration.

Example with Laddoo Problem:

Imagine you have 3 laddoos and 2 boxes, and you want to find all possible distributions of the laddoos into the boxes. You can use a stack to keep track of the current state of distribution and explore all possible configurations.

1. **Initial State**: Push the starting state (0 laddoos in both boxes) onto the stack.
2. **Exploration**: Pop a state from the stack, e.g., (0, 0), and generate new states by adding a laddoo to either of the boxes. For (0, 0), you might generate (1, 0) and (0, 1).
3. **Push New States**: Push these new states onto the stack.
4. **Backtrack**: If you reach a state where you can't distribute the remaining laddoos (e.g., all laddoos are distributed or no more boxes), backtrack by popping the state and exploring other options.

The stack helps in remembering where you are in the exploration and allows you to backtrack efficiently to previous states.

2)

The **call stack** is a stack data structure used by programming languages to manage function calls and their states. Each time a function is called, a new frame is pushed onto

the call stack with information about the function's parameters, local variables, and return address.

Example with Laddoo Problem: Consider a recursive function to find all possible distributions of n laddoos into k boxes.

1. **Recursive Call**: Each time distributeLaddoos(n, k) is called, a new frame with the current values of n and k is pushed onto the call stack.
2. **Base Case**: When n reaches 0 (no laddoos left) or k reaches 0 (no boxes left), the function reaches the base case and returns.
3. **Returning**: As the base case is hit, the function starts to return, popping frames from the stack and resuming the previous function calls.

The call stack keeps track of each recursive call's state and ensures that the program returns to the correct state after each function call completes.

In summary, stacks in backtracking algorithms manage state exploration and backtracking, while the call stack in recursion manages function calls and their states. Both concepts help in efficiently exploring and solving problems by managing different aspects of the computation process.