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

Batch: D2	**Roll No.: 16010122323**
Experiment / assignment / tutorial No. 4	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of Basic operations on stack using Array and Linked List- Create, Insert, Delete, Peek.

Objective: To implement Basic Operations on Stack i.e. Create, Push, Pop, Peek

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

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. Plate Stack in Buffet Restaurants: In a buffet restaurant, plates are stacked on top of each other. When a customer takes a plate, they take the one on top, which is the last one to be placed. As new plates are added, they are stacked on the top, and the plates at the bottom will be used last.

2. Stacking Books: When you stack books on a table or shelf, you place the last book you picked up at the top. When you want to access a specific book, you'll remove the topmost book first, following the Last in First out principle.

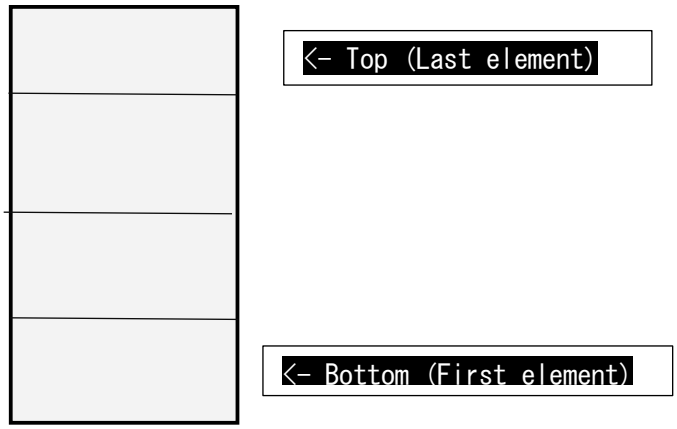
3. Browser Back Button: The browser's back button functionality is often implemented using a stack. As you browse different web pages, each page is pushed onto the stack. Pressing the back button pops the latest page from the stack, taking you back to the previously visited page.

4. Undo/Redo Operations in Software: Many software applications use a stack to manage undo and redo functionality. Each action you perform is pushed onto the stack, allowing you to undo by popping the last action and redo by pushing previously undone actions back onto the stack.

5. Call Stack in Programming: In computer programming, a call stack is used to keep track of function calls and their corresponding return addresses. When a function is called, it is pushed onto the call stack, and when the function completes, it is popped off the stack, ensuring that the program execution follows the Last in First out order. This is essential for maintaining the flow of execution in most programming languages.

Diagram:

STACK



Explain Stack ADT:
Abstract Data Types stores data and allow various operations on the data to access and change it.

It is a mathematical model, together with various operations defined on the model. An ADT is a collection of data and associated operations for manipulating that data

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

POP

1] Check Stack Underflow condition

If(top== -1)

Then print stack is underflow

2]Otherwise, delete the top position element

Popped_item=stack_arr[top]

3] Decrease the position of top

top=top-1

4] Print the popped_item

PUSH

1] Check Stack Full condition

If(top==MAX-1)

Then print stack is full or overflow

2] Otherwise increase the top value by 1

top=top+1

3] Input the value

4] Assign the item at top position stack_arr[top]=pushed_item

DISPLAY

Step 1: Check whether stack is EMPTY. (top == -1)
Step 2: If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
Step 3: If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
Step 3: Repeat above step until i value becomes '0'**Implementation Details:**

Built-In Functions/Header Files Used: (exit() etc)

#include<stdio.h>

#define max 10

Program source code:

#include<stdio.h>

#define max 10

int stack[max];

int top=-1;

void pop();

```
void push();
void display();
void peek();
int n,x,p,i;
int main(){
   void peek(){
      if(top==-1){
         printf("Stack underflow");
      }
      else{
         printf("%d\n",stack[top]);
      }
   }
   void push(){
      if (top>=max-1){
         printf("Stack overflow\n");
      }
      else {
         printf("Enter the value to be pushed\t");
         scanf("%d",&x);
         top++;
         stack[top]=x;
      }
   }
   void pop()
```

```
{
   if (top<=-1){
      printf("Stack underflow\n");
   }
   else {
      printf("popped item is \n%d\n",stack[top]);
      top--;
   }
}
void display()
{
   if (top>=0){
      printf("The stack is :\n");
      for (i=top;i>=0;i--){
         printf("\n%d\n",stack[i]);

      }
   }
}
while(n!=5){
printf("Enter choice from  1 to 4 for push, pop, display,peek and exit resp.\n");
scanf("%d",&n);
   switch(n)
{
   case 1:
```

```
    {
        push();
        break;
    }
case 2:
    {
        pop();
        break;
    }
case 3:
    {
        display();
        break;
    }
case 4:
    {
        peek();
        break;
    }
case 5:
    {
        printf("\n \t EXIT");
        break;
    }
default:
```

```
        {
            printf("Enter a valid case");
            break;
        }
    }
    }
}
```

Output Screenshots:

```
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
1
Enter the value to be pushed    6
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
1
Enter the value to be pushed    36
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
1
Enter the value to be pushed    22
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
3
The stack is :

==> 22

==> 36

==> 6
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
2
popped item is
22
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
4
==> 36
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
2
popped item is
22
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
4
==> 36
Enter choice from  1 to 4 for push, pop, display,peek and exit resp.
5
EXIT
```

Applications of Stack:

1. Expression Evaluation: Stacks are widely used in applications involving arithmetic expression evaluation. Infix expressions can be converted to postfix or prefix notation using stacks, which allows for efficient evaluation. Stacks help in maintaining the correct order of operations and handling operator precedence.

2. Function Call and Return: Stacks play a crucial role in managing function calls and returns in programming languages. Whenever a function is called, its return address and local variables are pushed onto the call stack. This allows the program to remember the point of execution and resume it once the function completes its execution.

3. Undo/Redo Operations: Stacks are commonly used in applications that require undo/redo functionality. Each action or operation is pushed onto the stack, allowing users to undo or redo their actions by popping and reapplying the operations in the reverse or forward order.

4. Balanced Parentheses and Syntax Parsing: Stacks are employed in checking the balance of parentheses, braces, and brackets in expressions. They can also be used in syntax parsing and validation, ensuring that the opening and closing symbols in a program are correctly matched and nested.

Explain the Importance of the approach followed by you.

1. Modularity: The code uses separate functions for each stack operation (push, pop, display, peek). This approach promotes code modularity, making it easier to understand, modify, and maintain the stack operations independently.

2. Code Reusability: The separate functions for stack operations make the code reusable. Other parts of the program or even different programs can easily utilize these functions to work with stacks without the need for rewriting the code.

3. Readability: The code is written in a straightforward manner, with each function having a clear purpose and functionality. This improves code readability, making it more accessible to developers, including those new to the codebase.

4. Error Handling: The code effectively handles potential stack underflow and overflow situations in the push, pop, and peek functions. It prints informative error messages to the user, preventing program crashes and enhancing user experience.

5. User Interaction: The code allows user interaction through the console, where the user can choose stack operations (push, pop, display, peek) using numeric choices. This approach provides a simple and intuitive interface for interacting with the stack.

6. Control Flow: The code uses a while loop to repeatedly prompt the user for the choice of stack operation until they choose to exit (option 5). This loop ensures a continuous user experience while handling different stack operations.

7. Constants and Variables: The code defines a constant "max" to set the maximum size of the stack and initializes a variable "top" to represent the current top index of the stack. These practices make the code more maintainable and allow for easy adjustment of the stack size.

8. Input Validation: The code includes a default case in the switch statement to handle invalid user input. This ensures that the program responds appropriately when the user enters a choice outside the valid range (1 to 5).

Conclusion: -

Thus, the Basic Operations on Stack i.e. Create, Push, Pop, Peek is implemented.

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] Stacks can be used in Backtracking algorithms to efficiently manage the state of the search space during exploration. When exploring a decision tree or a graph to find a solution, the current state and the potential future states are pushed onto the stack. If a dead-end is reached, backtracking involves popping the stack to return to the previous state and explore alternative paths. This process continues until a solution is found or all possible paths are explored.

Example: In the N-Queens problem, where N queens are to be placed on an N x N chessboard without attacking each other, a stack can keep track of the current state of queens' positions during the search for a valid solution. If a conflict arises, the algorithm backtracks by popping the stack and trying a different placement for the queens until a valid configuration is found.

2] In recursion, the call stack is a fundamental concept that represents the sequence of function calls made during the recursive process. Each recursive function call pushes a new frame onto the call stack, storing its local variables and return address. As the recursion unfolds, these frames are popped off the stack, allowing the program to return to previous function calls and continue execution until the base case is reached. The

call stack ensures proper function execution and memory management in recursive algorithms.