**Batch: A1 Roll No.: 1911004**

**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 & 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 –* EllisHorowitz,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*](https://www.cprogramming.com/tutorial/computersciencetheory/stack.html)
5. [*https://www.geeksforgeeks.org/stack-data-structure-introduction-program/*](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*](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).

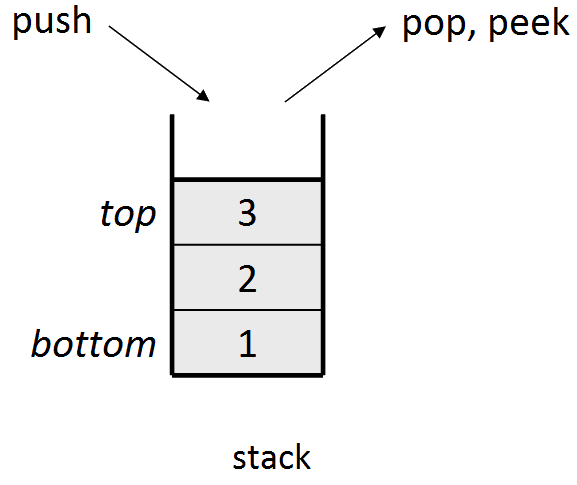
Students need to first try and understand the implementation of using arrays. Once comfortable with the concept, they can further implement stacks using linked list as well.

**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 operations in computer**
2. **Infix to postfix & vice versa**
3. **Stack of chairs & Stack of CDs in CD case**
4. **Reverse of a string is obtained by concept of LIFO of Stack**
5. **Stack of Bangles worn**
6. **Back/Forward on browsers are perform using stacks.**

**Diagram:** 

**Explain Stack ADT:**

**Stack is ADT(Abstract Data Type).It is used in most of the Programming Languages and also in Assembly Languages for Microprocessors.**

It has been given name STACK as the concept of STACK is based on REAL WORLD STACK. Example : **Stack of CDs in CD case.**

**Stack is based on idea of LIFO (Last In First Out). This can be explained as THE ITEM/ELEMENT WHICH COMES IN TO STACK FIRST WILL BE OBTAINED IN THE END(LAST)** & vice versa .

Aswe know that in real life example we can **remove or add** items only from **one end** same is with **COMPUTER STACK ADT**. This **One Side Open To** **ADD/DELETE** is **TOP OF STACK .** The **First item in makes the bottom of stack.**

**Thus at the given instance we can only work/perform operation on this element which is TOP OF STACK. In Stack ADT :**

* **addition of item/element is PUSH**
* **deletion is POP**

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

* **ARRAY :**

1. **Start**
2. **Display OPTIONS**
3. **creation :**

* **int stack[100]; // create a stack having max elements = 100**

1. **insertion/push :**

* **input value to be pushed;**
* **pass value in to the stack by invoking &paasing it to push function**
* **if top==max-1 : print ‘stack overflowed’;**
* **else : top++ ; assign stack[top]=val;**

1. **deletion/pop :**

* **if top==-1 : print ‘stack is underflowed’ ;**
* **else : value =stack[top] is returned ; top-- ;**

1. **displaying :**

* **display() function is called**
* **if : top==-1 : print stack is empty**
* **else :**
  + **repeat 1st to 3rd till (i>=0)**
  + **print stack[i];**
  + **i--;**

1. **End**

* **LINKLIST**

1. **Start**
2. **Display OPTIONS**
3. **push/insert :**

* **memory allocation using malloc() ;**
* **if : ptr==null : link-list ptr is null thus can not push element;**
* **else : input element;**
* **if head == null : first element is added to the linked list**
* **else element is added before the head element**
* **print element pushed on to stack;**

1. **pop/delete:**

* **if head ==null : print “underflow” ;**
* **else : head element is stored in item ; ptr=head, head=head->next;**
* **free(ptr ) ;**
* **element popped & print message ‘POPED’;**

1. **display :**

* **if (ptr==null) : print “stack empty”;**
* **else follow below steps until ( ptr!=NULL)**
  + **print ptr->val**
  + **ptr=ptr->next**

1. **destroy :**

* **In this pop operation is carried repeatedly**
* **if (ptr==null) : print “stack empty”;**
* **else : while(ptr!=null):**

**item = head->val;**

**ptr = head;**

**head = head->next;**

**free(ptr);**

1. **End**

**Implementation Details:**

* **Enlist all the Steps followed & various options explored:**

**In both the approaches a Menu Driven program is written in which the options are given , that is (push, pop, display, destroy, peep, exit, default) and the user enters the option.**

**Respective messages are shown on completion of operation for every selected choice. In each case the function of the respective operations are invoked & at the end of the case, a break statement is executed. FOR**

* **push() the function takes in value ;**
* **pop () return poped value**
* **peep() return peeped value**
* **destroy( ) & display () destroy & display stack respectively**
* **exit helps to break do while & switch case**
* **Assumptions made for Input:**

For this particular example we assume that we would take only **INTEGER VALUES IN TO THE STACK** for **STACK ADT OPERATION .**

So we have created **int[]** array for **implementation stack.**

* **Built-In Functions Used: malloc() free()** in link-list implementation
* **Program source code:**

1. **ARRAY**

**#include <stdio.h>**

**int max\_size=100 , top=-1;**

**int Stack[100];**

**int isempty() {**

**if(top == -1)**

**return 1;**

**else**

**return 0;**

**}**

**int isfull() {**

**if(top == max\_size)**

**return 1;**

**else**

**return 0;**

**}**

**void push(int pushOn){**

**if(!isfull()) {**

**top++;**

**Stack[top] = pushOn;**

**//return top;**

**printf("PUSHED %d on to stack\n",pushOn);**

**} else {**

**printf("-->Could not insert data, \n 'Stack is full.'\n");**

**} //return 0;**

**}**

**void peep(){**

**if (top<=-1){**

**printf("\nEmpty Stack\n");**

**}else{**

**printf("Element at top of the stack: %d\n" ,Stack[top]);**

**}**

**}**

**int pop() {**

**int data;**

**if(!isempty()) {**

**data = Stack[top];**

**top --;**

**return data; //printf("POPED %d",data);**

**} else {**

**printf("-->Could not retrieve data, \n 'Stack is empty.'\n");**

**}**

**}**

**void display(){**

**if(top>=0){**

**printf("\nElement of Stack are : \n");**

**for (int i=top;i>=0;i--){**

**printf("%d ",Stack[i]);**

**}**

**}else if (top<=-1 || top<0){**

**printf("\nEmpty Stack");}**

**}**

**void destroy(){**

**int poped;**

**if (top>-1){**

**printf("\nElement of Stack are : \n");**

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

**poped=pop();**

**} printf("\nStack Destroyed\n");**

**}else{**

**printf("\nEmpty Stack\n");**

**}**

**}**

**int main(void) {**

**int option,pushOn,popOut,top,peeped;**

**printf("STACK IMPLEMIMPLEMENTATION");**

**do{**

**printf("\n1.Push\n2.Pop\n3.Peep\n4.Destroy\n5.Display\n0.EXIT\n");**

**scanf("%d",&option);**

**switch(option){**

**case 1: printf("\nEnter number to be pushed onto stack: ");**

**scanf("%d",&pushOn); push(pushOn);**

**break;**

**case 2: popOut=pop();**

**printf("\nNumber to be poped out of stack: %d",popOut);**

**break;**

**case 3: peep();**

**break;**

**case 4: destroy();**

**break;**

**case 5: display();**

**break;**

**case 0: printf("EXIT");**

**break;**

**default: printf("\nWRONG OPTION!");**

**}}while(option!=0);**

**return 0;**

**}**

1. **LINK LIST**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int val;**

**struct node \*next;**

**};**

**struct node \*head;**

**int push() {**

**int val;**

**struct node \*ptr = (struct node\*)malloc(sizeof(struct node));**

**if(ptr == NULL) {**

**printf("Not able to push the element 'overflow' \n");**

**} else {**

**printf("Enter the value ");**

**scanf("%d",&val);**

**if(head==NULL) {**

**ptr->val = val;**

**ptr -> next = NULL;**

**head=ptr;**

**} else {**

**ptr->val = val;**

**ptr->next = head;**

**head=ptr;**

**}**

**printf("Item pushed\n");**

**}**

**return 0;**

**}**

**int pop() {**

**int item;**

**struct node \*ptr;**

**if (head == NULL) {**

**printf("Underflow\n");**

**} else {**

**item = head->val;**

**ptr = head;**

**head = head->next;**

**free(ptr);**

**printf("Item popped\n");**

**} return 0;**

**}**

**int display() {**

**struct node \*ptr;**

**ptr=head;**

**if(ptr == NULL) {**

**printf("Stack is empty\n");**

**} else {**

**printf("Printing Stack elements \n");**

**while(ptr!=NULL) {**

**printf("%d\n",ptr->val);**

**ptr = ptr->next; }**

**} return 0;**

**}**

**void destroy(){**

**int item;**

**struct node \*ptr;**

**if(ptr == NULL) {**

**printf("Stack is empty 'Underflow'\n");**

**}else{ printf("Stack Destoyed\n");**

**while(ptr!=NULL){**

**item = head->val;**

**ptr = head;**

**head = head->next;**

**free(ptr);**

**} printf("Stack Destoyed\n"); }**

**}**

**void peep(){**

**int v;**

**struct node \*ptr;**

**ptr=head;**

**if(ptr == NULL) {**

**printf("Stack is empty\n");**

**} else {**

**printf("Printing Stack element at top \n");**

**printf("%d\n",ptr->val);**

**ptr = ptr->next; }**

**}**

**int main () {**

**int choice=-1;**

**printf("\n\t=== Stack operations using linked list ===\n");**

**while(choice != 0) {**

**printf("\nChoose one from the below options...\n");**

**printf("\n1.Push\n2.Pop\n3.Display\n4.Destroy\n5.Peep\n0.Exit");**

**printf("\n Enter your choice \n");**

**scanf("%d",&choice);**

**switch(choice) {**

**case 1: push();**

**break;**

**case 2: pop();**

**break;**

**case 3: display();**

**break;**

**case 4: destroy();**

**break;**

**case 5: peep();**

**break;**

**case 0: printf("\t\t\tEXIT");**

**break;**

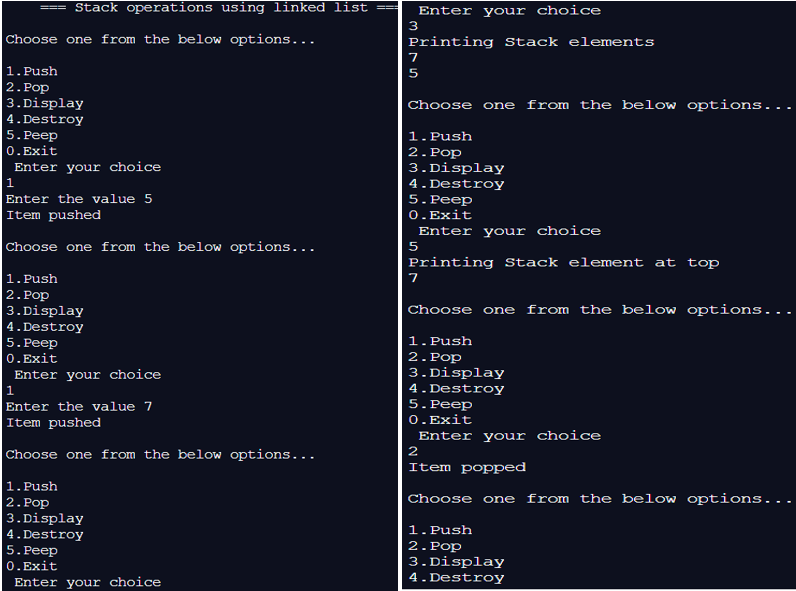
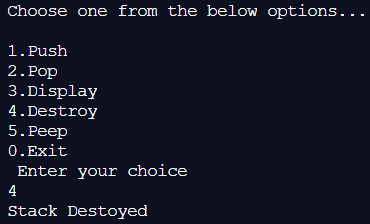
**default:**

**printf("Please Enter Valid option: ");**

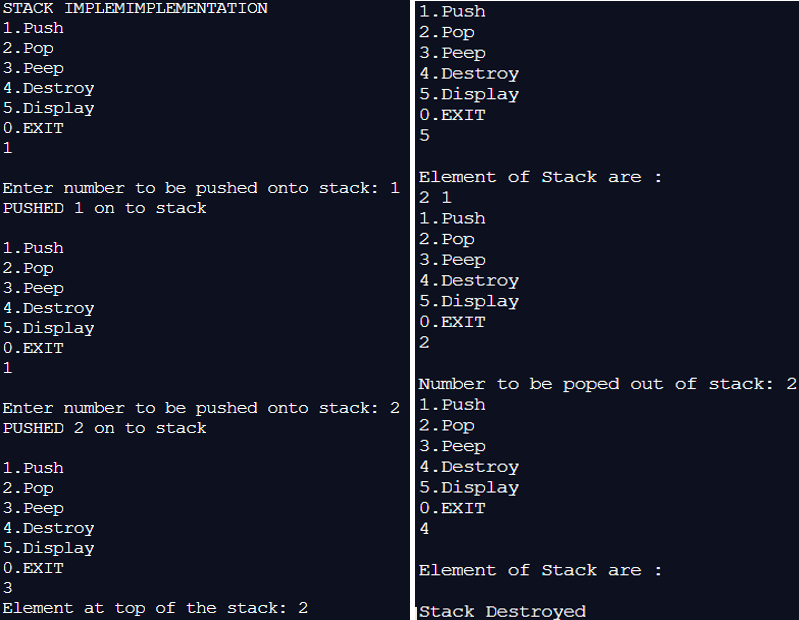
**}; } return 0;**

**}**

**Output Screenshots: Link List**

 **1**  **2**

**Array**



* **Applications of Stack:**

1. **Undo/Redo in Computers**
2. **Backtracking**
3. **Reversing a String**
4. **Infix to Postfix & vice versa**
5. **Recursion**
6. **Back/Forward on browsers are perform**
7. **Storage of data & opcodes in Assembly language Programming**

* **Explain the Importance of the approach followed by you**
* **Learning working of STACKS ADT with LINKED LIST /ARRAY**
* **Using the arrays to implement LIFO (FILO (STACK ADT )) & their concepts**
* **Dynamically allocate memory using malloc()**
* **Viewing memory allocation to a stack by linked list**
* **Operations like push , pop, delete ,peep ,destroy**
* **Importance & use Of Stack ADT**
* **Conclusion:** Through this experiment the concept of **Stack ADT** along with is **idea of LIFO/FILO** was understood. **Creation/deletion & memory allocation of Stack with Array /Link List** was understood along with Stack Applications & Use.
* **Post Lab Questions:**

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

**ANS:**

**Backtracking is an algorithmic-recursive-technique for solving problems by trying to build a solution incrementally(updating), one value at a time, thus by removing those solutions that fail to obey the constraints of the given problem at any point of time. An important application of stacks is backtracking .**

**Consider a simple example of finding the correct path in a maze. There are a series of points, from the starting point to the destination. We start from one point. To reach the final destination, there are several paths. Suppose we choose a random path. After following a certain path, we realise that the path we have chosen is wrong.**

**So we need to find a way by which we can return to the beginning of that path. This can be done with the use of stacks. With the help of stacks, we remember the point where we have reached. This is done by pushing that point into the stack. In case we end up on the wrong path, we can pop the last point from the stack and thus return to the last point and continue our quest to find the right path.**

1. **Illustrate the concept of Call stack in Recursion.**

**ANS:**

**In Recursion ,the function calls it self many time to perform particular task / calculations. The function CALLS ITSELF REPEATEDLY BY UPDATING THE ARGUMENT PARAMETERS ACCORDINGLY till the BASE CONDITION WHICH IS ALWAYS OUR END CONDITION IS FOUND TRUE .**

**For function calls to stop & to get desired output this BASE CONDITION is needed . After which return values takes it back to 1st call. It’s the bottom of the barrel (stack(LIFO)), the end of the recursive calls.**

**The result of the recursion goes back to the top of the call stack, it is CALL STACK in the computer that keeps up the record of all the functions invoked in a program. Thus programming languages have CALL STACKS are important to recursion.**