**Batch: A3 Roll No.: 1911034**

**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

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)

**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.

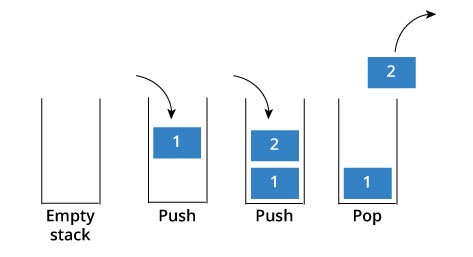
**Related Theory: -**

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:**

* A **stack of books** in a cupboard.
* Wearing/Removing **Bangles**.
* Support for**recursion (**Activation records of method calls).
* A good real-life example of a stack is the pile of dinner plates that we encounter when we eat at the local cafeteria: When we remove a plate from the pile, we take the plate on the top of the pile. But this is exactly the plate that was added ("inserted'') most recently to the pile by the dishwasher. If we want the plate at the bottom of the pile, we must remove all the plates on top of it to reach it.
* Java Virtual Machine (JVM) uses stack
* Arithmetic Expression Evaluation.

**Diagram:**



**Explain Stack ADT:**

The stack data structure can be defined as an ADT in terms of the operations that can be performed of the stack. A stack is an ordered collection of elements , where the insertion as well as deletion takes place at only one end , which is called as the ‘top’. Some of the operations that can be performed on the stack are as follows :

1. push() : insert an element on top of the stack
2. pop() : delete an element from the stack
3. peek(): view the topmost element of the stack
4. display(): display the entire stack , starting from the topmost element.

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

**For stack data structure using arrays**

1. **Algorithm for creation of stack**

**Step1:** Define MAX that stores the maximum number of variables that a stack as an array can hold.

**Step2:** Create a struct called node that contains 2 elements, an integer variable named top , and an array of MAX size that holds all the elements.

**Step3**: In the function main(), create a struct variable that will represent the stack for storing the elements and a pointer to hold the address of the stack to be passed between different functions

**Step4**: In main(), initialize the value of top =-1.

1. **Algorithm to insert an element in the stack (push operation)**

Step1**:** If TOP = MAX-1;

Print “Stack is completely full”

Go to step 4

[END OF IF]

Step2: TOP = TOP+1

Step3: St[TOP]= data

Step4: END

(note – here data is the value of the element by the user)

1. **Algorithm to display the stack** :

Step 1: If TOP= -1

Print(“Stack is empty”)

Go to step

Step 2: initialize a variable PTR which holds the top address

Step 3: repeat step 4 while PTR!=NULL

Step 4: print( “PTR-> DATA)

Step 5: Exit

**Algorithm to pop an element off the stack**

Step 1: Create a variable named PTR which holds the address of the top element

Step 2: Set TOP = TOP-> NEXT( such that it points to the next element which is now the new top element of the linked list)

Step 3: Free PTR

**Algorithm for Stack Operations using Linked List**

**Algorithm to insert the element in the stack using Linked list :**

Step1: Allocate memory for the new node by creating a pointer (ptr) of stack type and using malloc function.

Step2: set ptr->data= val

Step 3: If top = NULL

Set ptr->next = NULL

Set top = ptr

ELSE

Set ptr-.next = top

Set top = ptr;

[END OF IF ]

Step 4: END

**Algorithm to delete an element off a linked stack**

Step 1: If top = NULL

Print (“Stack is empty”)

Go to step 5

[END OF IF]

Step 2: Set ptr = top

Step 3: Set Top = Top-> next

Step 4 : Free PTR

Step 5: Exit

**Implementation Details:**

1. **Enlist all the Steps followed and various options explored.**

In both the codes ,a menu driven program is followed which allows the user to enter the option of his / her choice in order to push / pop / peek an element.

The control is then transferred to the respective functions after getting the value from the user.

In the functions , the memory is allocated/ freed accordingly depending on the operation to be performed.

The control is then returned back to the main function , where the user can choose to perform more operations or can choose to exit

**Assumptions made for Input:**

In both the codes it is assumed that the data entered by the user is an integer value.

In the linked list , the data is assumed to be int , while in the array implementation , an integer array is created.

**Built-In Functions Used:**

**Free():** Used to free the space occupied by the node that is to be deleted

**Malloc()** used to allocate memory dynamically

**Program source code:**

**Implementation using array**

#include <stdio.h>

#define MAX 5

struct Stack

{

int top;

int a[MAX];

};

int push(struct Stack\*);

int pop(struct Stack\*);

int peek(struct Stack\*);

int display(struct Stack\*);

int main(void) {

struct Stack s;

struct Stack \*a=&s;

s.top=-1;

int c;

do {printf("\nEnter your choice\n");

printf("\n1 to push");

printf("\n2 to pop");

printf("\n 3 to peek");

printf("\n 4 to display");

printf("\n -1 to exit");

scanf("%d",&c);

switch (c)

{

case 1 :

{

push(a);

}

break;

case 2 :

{

pop(a);

}

break;

case 3 :

{

peek(a);

}

break;

case 4:

{

display(a);

}

}

}

while(c!=-1);

}

int push( struct Stack \*b)

{int val=0;

if(b->top==MAX-1)

{

printf("Stack is completely full");

}

else

{

printf("\n Enter the no. to be pushed on the stack");

scanf("%d",&val);

b->top=b->top+1;

b->a[b->top]=val;

//since we only pass the pointer to the struct , thus any object of the struct that we reference has to be incremented with the help of the pointer itself

}

}

int pop(struct Stack \*f)

{

if(f->top!=-1)

{

printf("The value popped off the stack will be %d",f->a[f->top]);

f->top--;

}

else

{

printf("Stack is already empty");

}

}

int peek(struct Stack \*d)

{

if(d->top!=-1)

{

printf("The element at the top is %d",d->a[d->top]);

}

else printf("The stack is empty");

}

int display(struct Stack \*e)

{int temp;

if(e->top!=-1)

{temp=e->top;

while(temp!=-1)

{

printf("\n%d",e->a[temp]);

temp--;

}

}

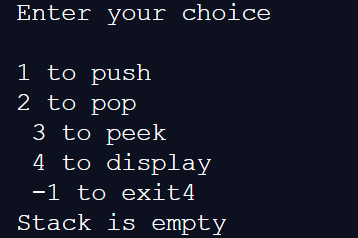
else

printf("Stack is empty");

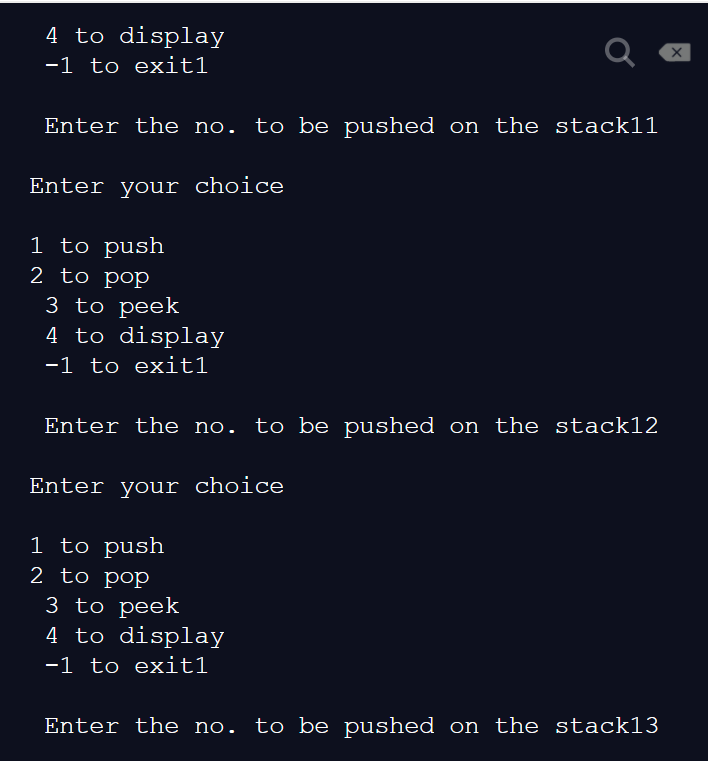
}

**Output Screenshots (linked list using array)**

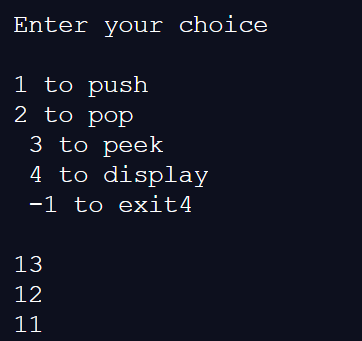
Case 1: Displaying the stack when it is empty (no numbers on the stack):



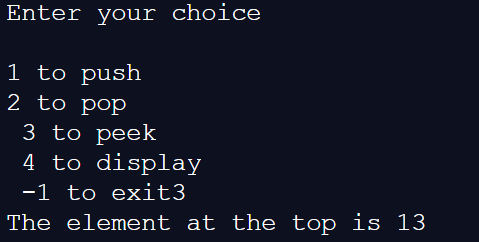
Case 2: pushing few numbers on the stack:



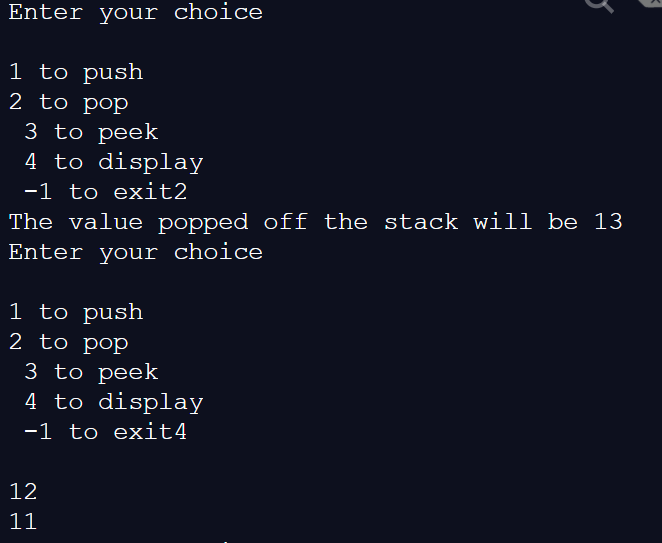
Case 3 : displaying the stack:



Case 4: peeking of stack (shows the topmost element):

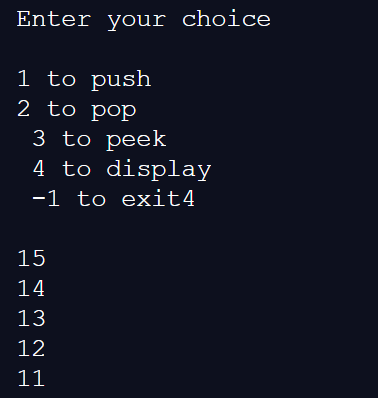


Case 5: popping of a number from the stack and displaying the stack after popping :

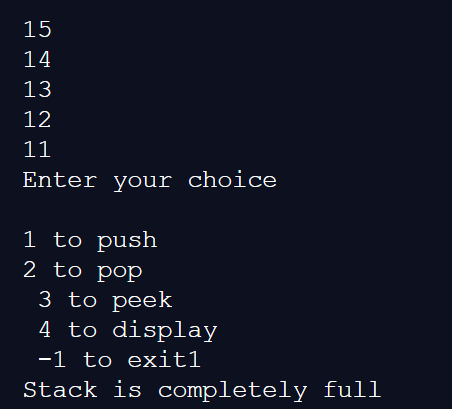


Case 6: after filling stack to MAX capacity(which is 5 in this case) , it will show that no more numbers can be entered :

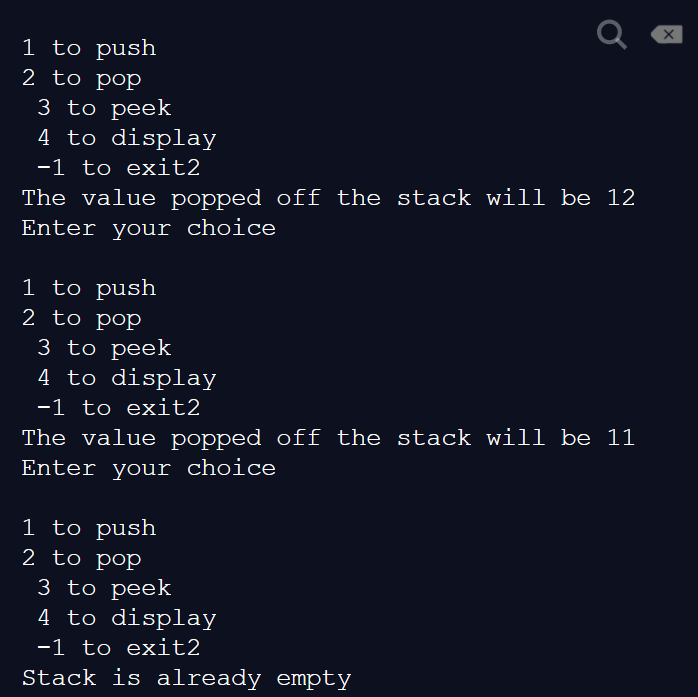
For this , we first fill the stack to its max capacity and then display the stack :



Now we cannot enter any more numbers on the stack as it is already full:



Last case: popping off numbers from an empty stack :



**Code for implementation using Linked List:**

#include <stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*next;

};

struct node\* push(struct node\*top, int val);

struct node\* pop(struct node \*top);

int peek(struct node \*top);

struct node\* display(struct node\*top);

struct node\* destroy(struct node \*top);

int main(void) {

struct node \*top = NULL;//used to hold the address of the node which is currently on the top

int c=0,val=0;

int temp;

do {printf("\nEnter your choice\n");

printf("\n1 to push");

printf("\n2 to pop");

printf("\n 3 to peek");

printf("\n 4 to display");

printf("\n 5 to destroy");

printf("\n -1 to exit");

scanf("%d",&c);

switch (c)

{

case 1 :

{

printf("\n Enter the no. to be pushed on the stack");

scanf("%d",&val);

top =push(top,val);

}

break;

case 2 :

{

top=pop(top);

}

break;

case 3 :

{

temp= peek(top);

if(temp==-1)

{

printf("The stack is empty");

}

else printf("The element at the top is %d", temp);

}

break;

case 4:

{

top =display(top);

}

break;

case 5:

{

top = destroy(top);

if (top==NULL)

{

printf("\n Stack has been destroyed");

}

}

}

}

while(c!=-1);

return 0;

}

struct node\* push(struct node\*top, int val)

{

struct node \*ptr;// used to hold the address of the new node every time it is created

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

ptr->data=val;

if(top==NULL)

{

ptr->next=NULL;

top=ptr;

}

else{

ptr->next=top;

top=ptr;

}

return top;

}

struct node\* pop(struct node \*top)

{struct node \*ptr;

ptr=top;

if(top==NULL)

{

printf("Stack is empty");

}

else

{

printf("The number popped off the stack is %d",ptr->data);

top=top->next;

free(ptr);

}

return top;

}

int peek(struct node\*top)

{

if(top==NULL)

{

return -1;

}

else return top->data;

}

struct node\* display(struct node\*top)

{struct node \*ptr;

ptr=top;

if(top==NULL)

{

printf("Stack is empty");

}

else

while(ptr!=NULL)

{

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

ptr= ptr->next;

}

return top;

}

struct node \*destroy(struct node\*top)

{

if (top==NULL)

{

printf("Stack is empty\n");

}

else while (top!=NULL)

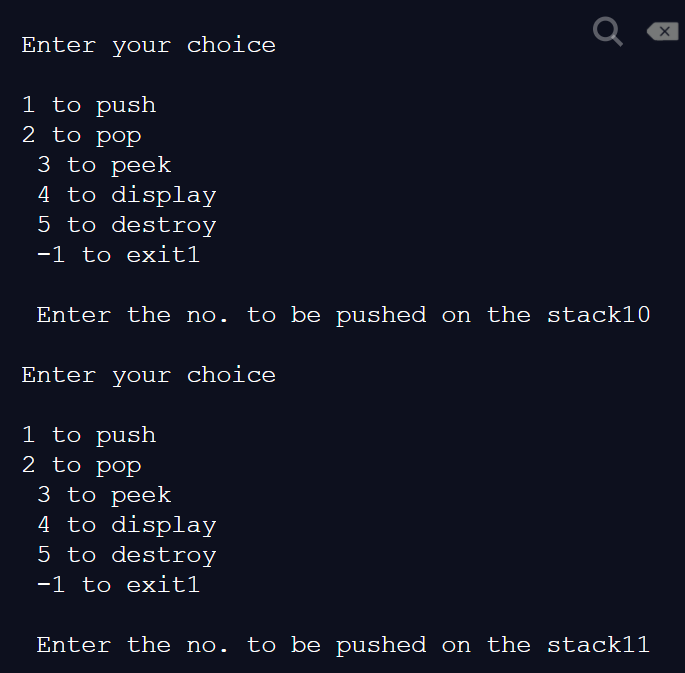
{

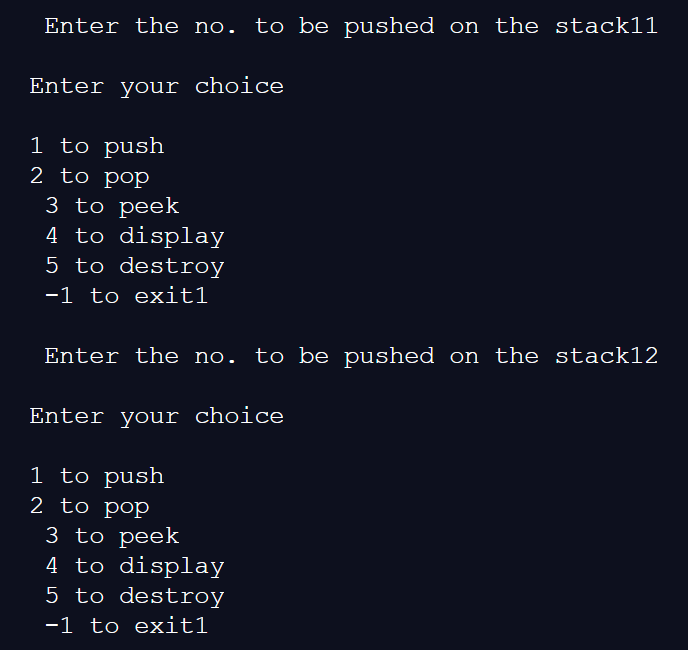
top = top->next;

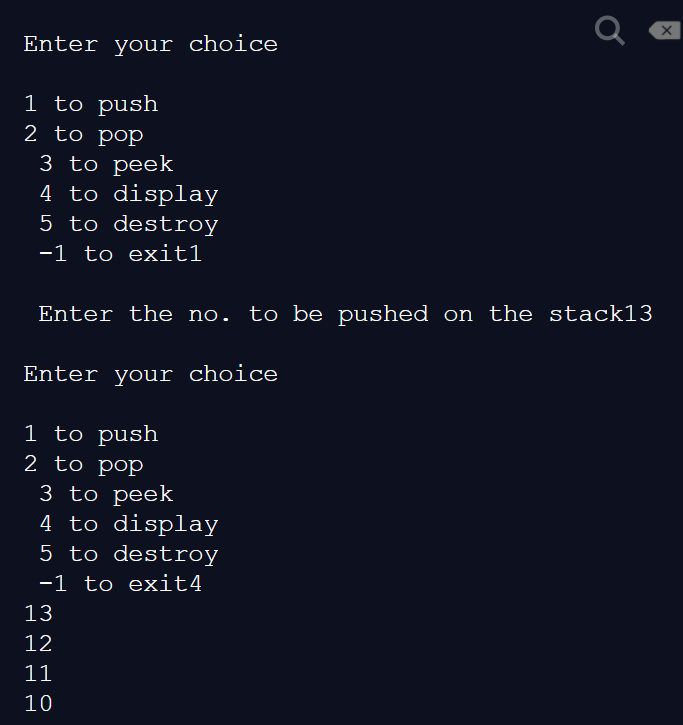
}

return top;

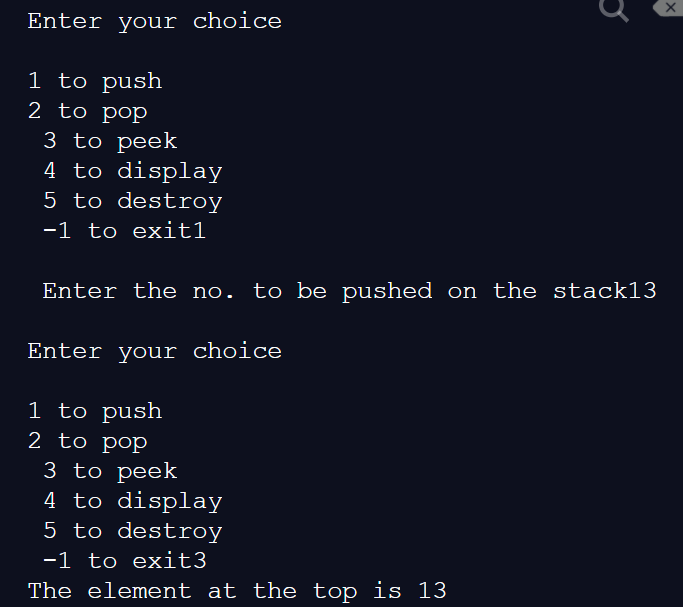
}











**Applications of Stack:**

The Stack is Last In First Out (LIFO) data structure. This data structure has some important applications in different aspect. These are like below −

1. Expression Handling −
   * 1. Infix to Postfix or Infix to Prefix Conversion −

The stack can be used to convert some infix expression into its postfix equivalent, or prefix equivalent. These postfix or prefix notations are used in computers to express some expressions.

* + 1. Postfix or Prefix Evaluation −

After converting into prefix or postfix notations, we have to evaluate the expression to get the result. For that purpose, also we need the help of stack data structure.

1. Stacks are also useful in the function call-and-return process. When we make a call from one function to another function , for some intermediate step, the address of the calling function is stored on the stack , and after the execution of the called function is completed, in order to return to the calling function , the address is popped off from the stack and control resumes from the point where the function was called
2. **String Reversal** The stack data structure is very useful in reversing a string.
3. **Syntax Parsing**

Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code.

1. Stack data structure is also very useful in the checking of parantheses in a given expression.

.

**Explain the Importance of the approach followed by you**

Through this approach , we have understood the stack as an Abstract Data Structure, that is with the operations like push , pop ,peek and display.

The stack data structure can be implemented using two ways, arrays and linked list , both of which have been understood in this experiment.

We have also used pointers which holds the address of the stack , and passed it between different functions for implementing changes to the stack data structure .

The special cases , such as when there occurs an overflow error , when the stack is completely full while using the push operation or an underflow error when the stack is empty during peek or pop operation has also been considered.

The uses of Stack data structure in various real life problems such as Backtracking Problem or N Queens problem has also been learnt.

.

**Conclusion:-**

Through this experiment , we have learnt about stack as an abstract data structure , the various operations that can be performed using stack as well as we have learnt , how to implement stack using linked list and array.

We have also learnt the application of stacks in real life.

**PostLab 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 (by time, here, is referred to the time elapsed till reaching any level of the search tree).

**Use of stack data structure in backtracking :**

**1.**Backtracking is a rather typical recursive algorithm, and any recursive algorithm can be rewritten as a stack algorithm. Recursive algorithms are translated into machine or assembly language.

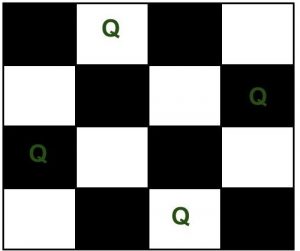
**2.** Starting from the root, the only nodes that can be pushed onto the stack are the children of the node currently on the top of the stack, and these are only pushed on one child at a time; hence, the nodes on the stack at all times describe a valid path in the tree. Nodes are removed from the stack only when it is known that they have no goal nodes among their descendents. Therefore, if the root node gets removed (making the stack empty), there must have been no goal nodes at all, and no solution to the problem.

**3.** When the stack algorithm terminates successfully, the nodes on the stack form (in reverse order) a path from the root to a goal node.

**Example :**

**N-Queens Problem-**

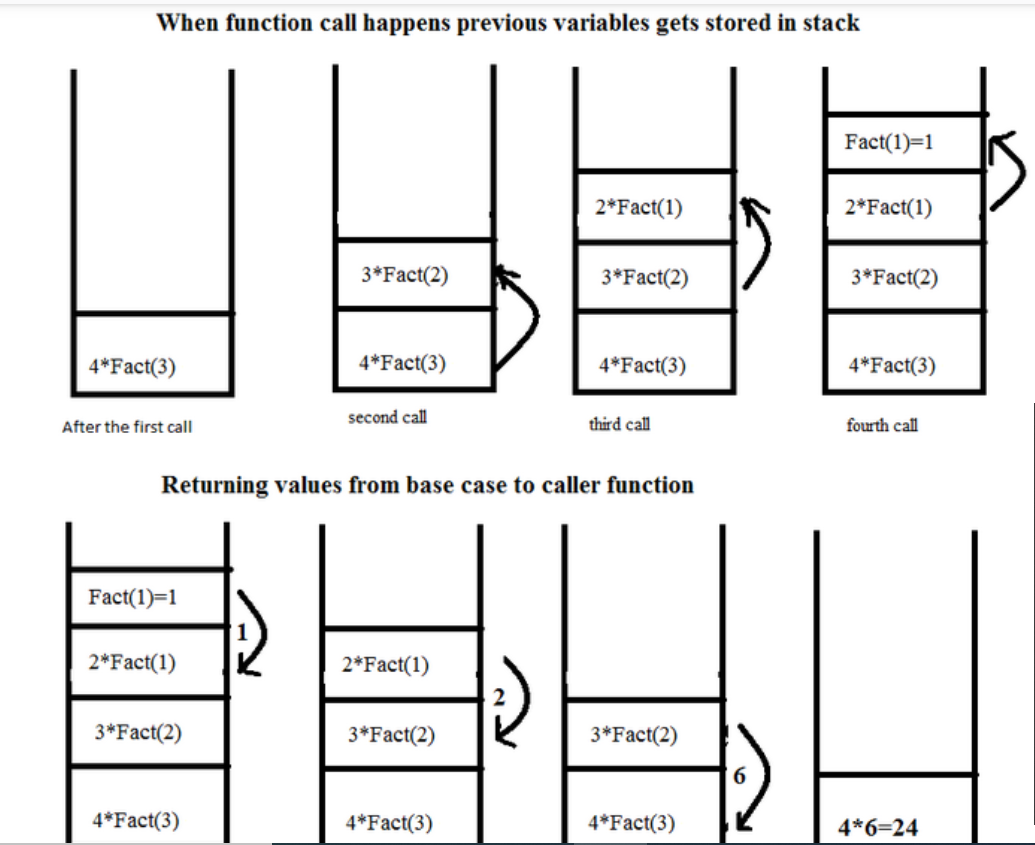
The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



The expected output is a binary matrix which has 1s for the blocks where queens are placed.

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

**The concept of call stack in recursion can be explained with the help of a simple example as follows :**



**The above example illustrates using the stack data structure for recursive function call. In this , we attempt to calculate the value of 4! using a recursive factorial function.**

**The algorithm works as follows :**

* + 1. When the program starts execution , the function factorial calculates 4! as 4x3! and sends a function call to calculate the value of 3!.
    2. This value of 3! will have to be returned to the calling function , hence the address of the calling function is stored on top of the stack.
    3. Meanwhile in order to calculate 3!, we would need the value of 2!, and so on.
    4. The base case is defined for x=1 , which is defined to be 1.
    5. After the value of 1! Is calculated , it is returned to the calling function of 2! , which is popped off from the stack. The value of 2! , is then returned to the calling function of 3! . Similarly after each instance of the program finishes execution , the address is returned to the calling function and that instance is popped off from the stack.
    6. When the last instance is popped off from the stack , the value is finally printed to the user.