**Batch: B3 Roll No.: 121**

**Experiment / assignment / tutorial No.**

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

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

|  |
| --- |
| **Title:**  Implementation of Basic operations on stack for the assigned application using Array and Linked List- Create, Insert, Delete, Peek. |

**Objective:** To implement Basic Operations on Stack i.e. Create, Push, Pop, Peek for the given application

**Expected Outcome of Experiment:**

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

**Books/ Journals/ Websites referred:**

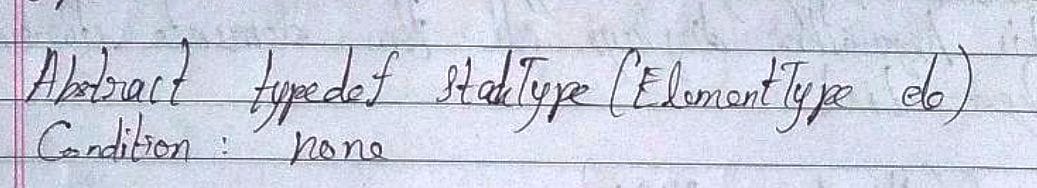
1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

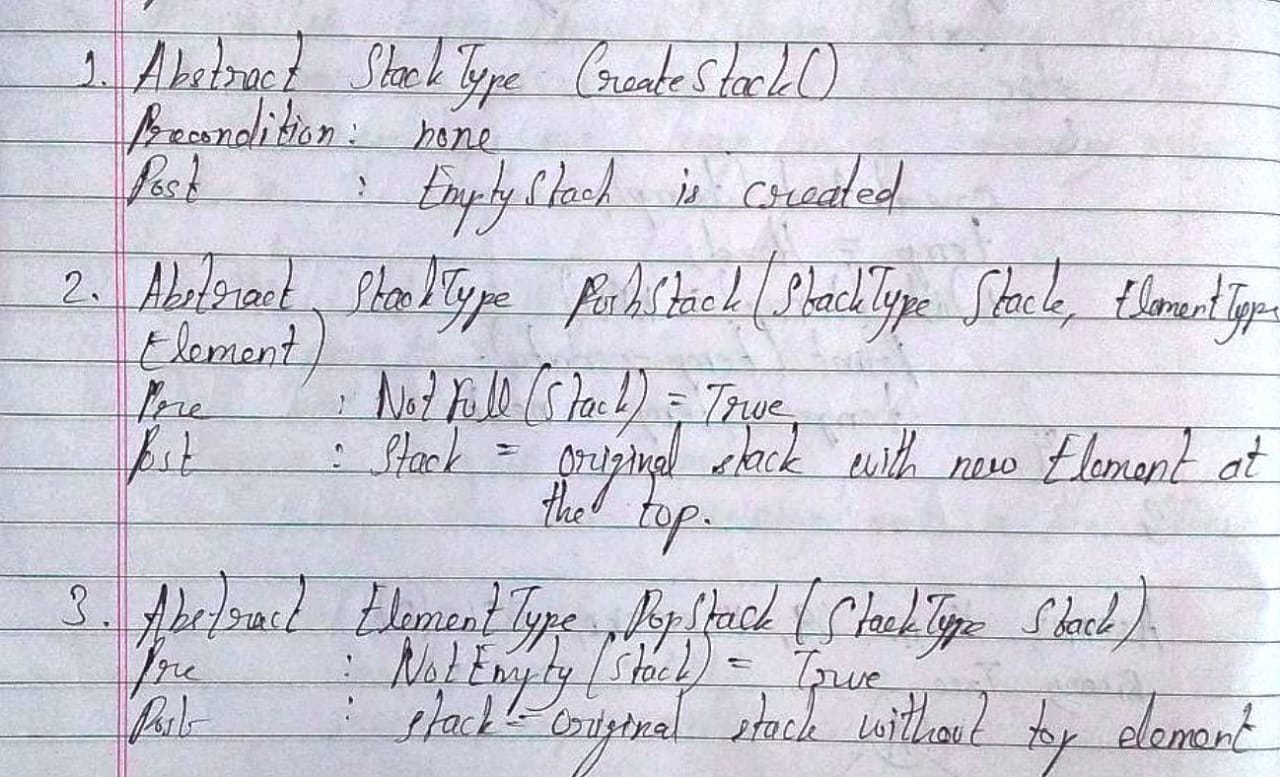
Paul G. Sorenson

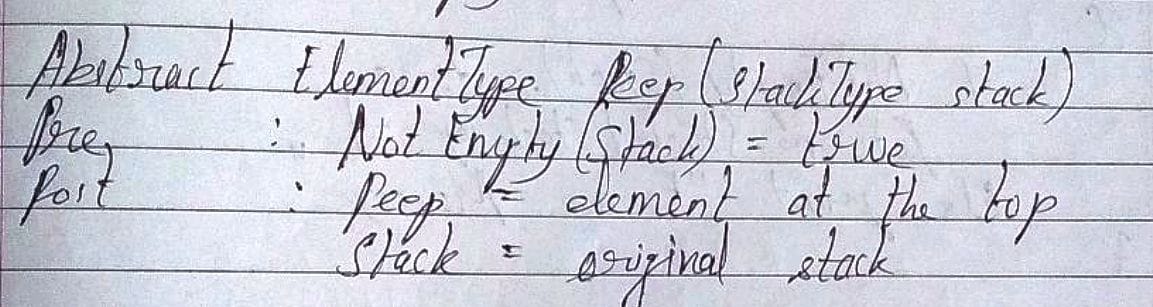
1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. [*https://www.cprogramming.com/tutorial/computersciencetheory/stack.html*](https://www.cprogramming.com/tutorial/computersciencetheory/stack.html)
3. [*https://www.geeksforgeeks.org/stack-data-structure-introduction-program/*](https://www.geeksforgeeks.org/stack-data-structure-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**:

(Define stack, enlist stack operations).







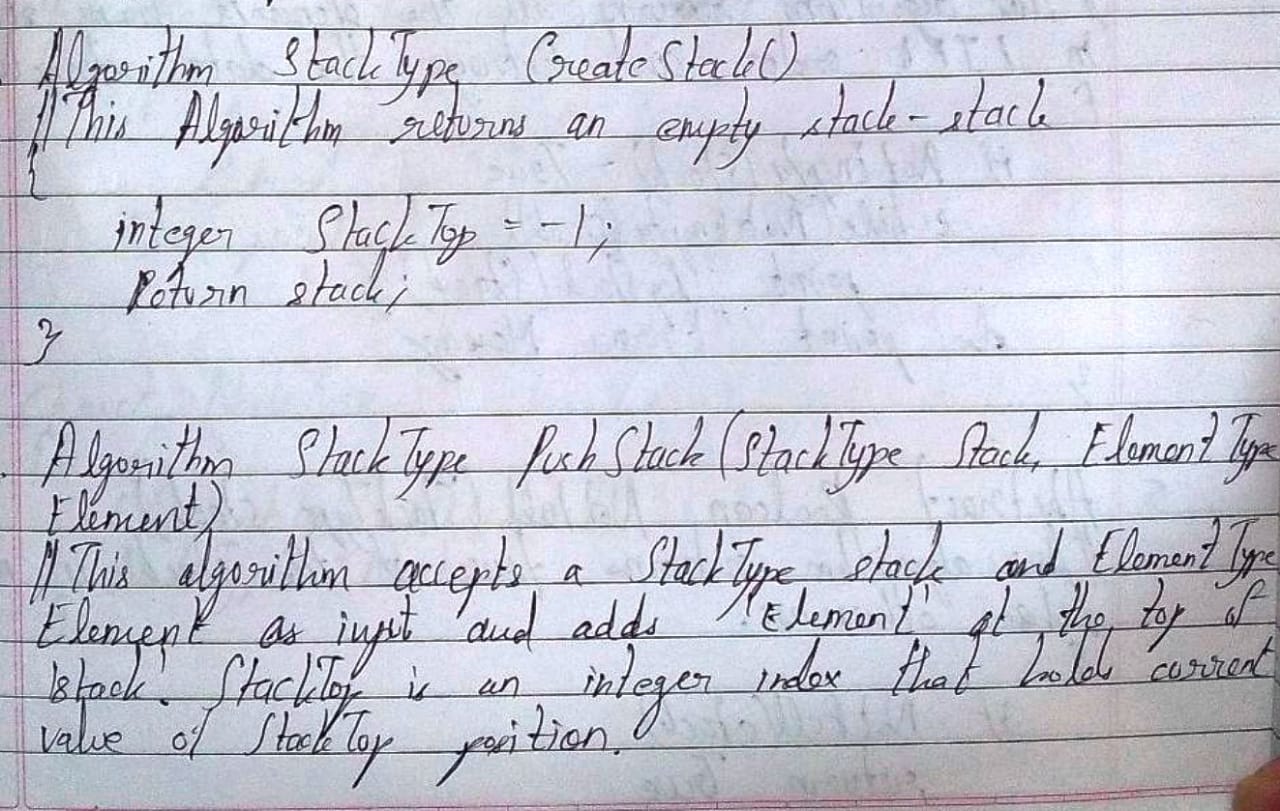
**List 5 Real Life applications:**

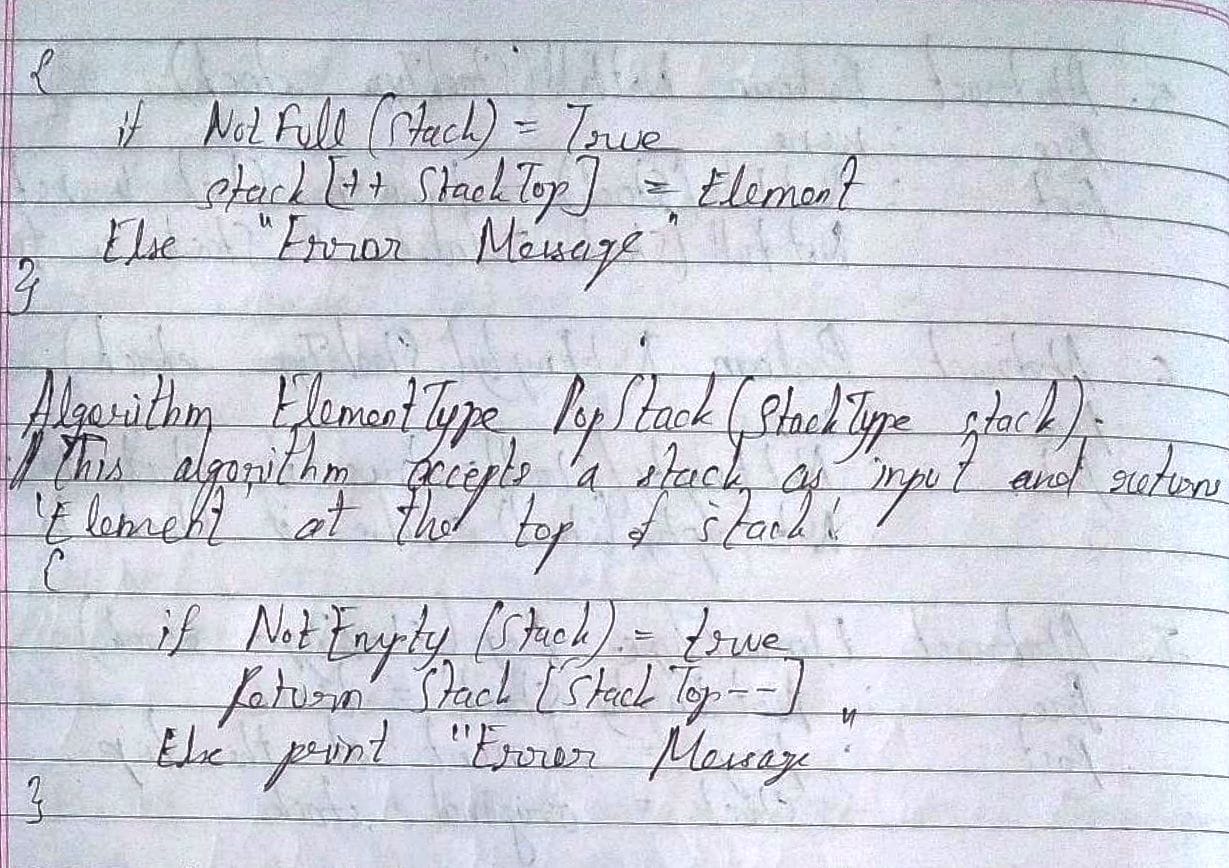
* + - 1. To store history in a web browser
      2. Undo operation in any application or text editor
      3. Saving local variables during function calls
      4. To test whether a string or a number is palindrome
      5. Parenthesis Matching Algorithm is based on the principle of stack and is used by compilers in IDE’s to check if the code has got proper placement of brackets.

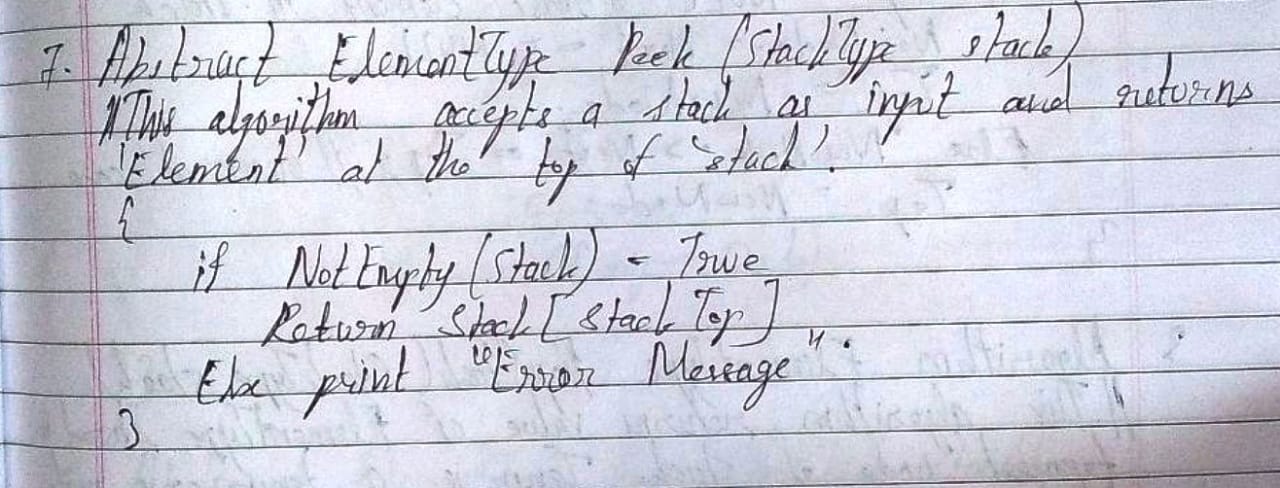
**Stack ADT:**

**Algorithm for stack operations using array/Linked list :**

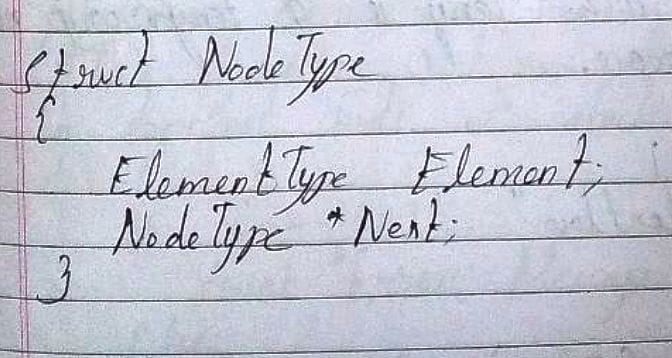
**Array Implementation:**

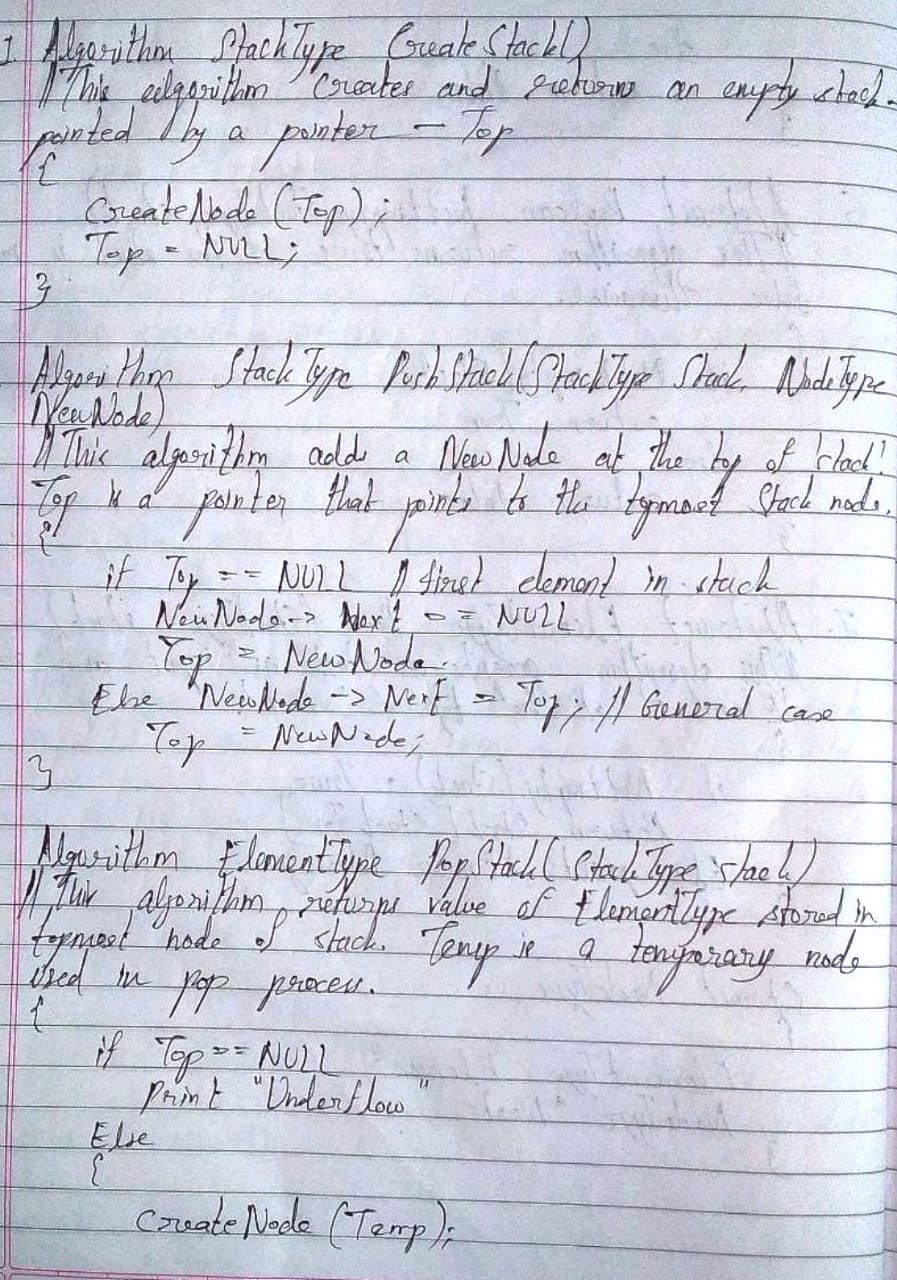
****

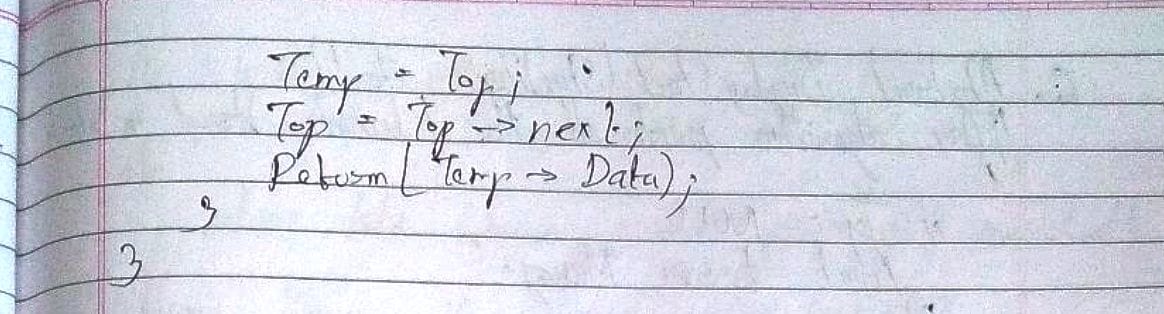
****

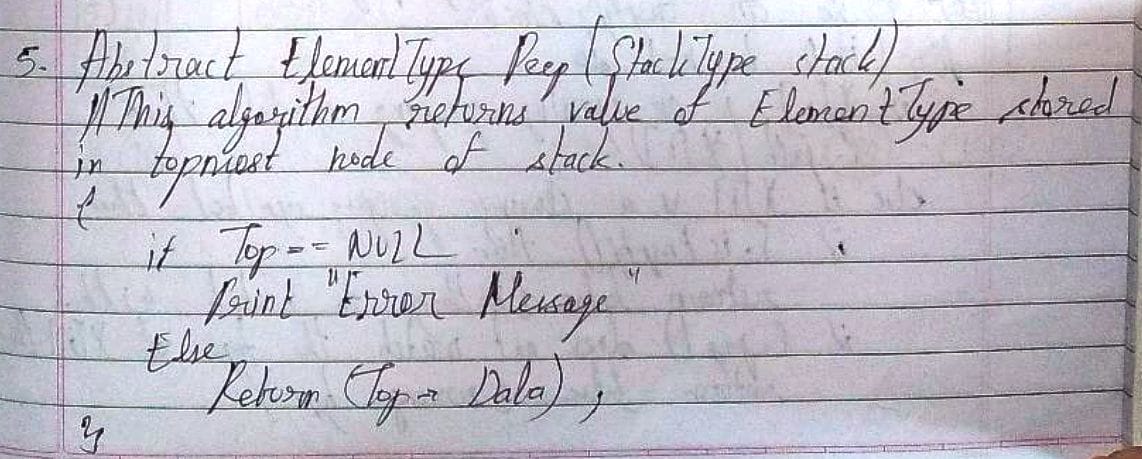
****

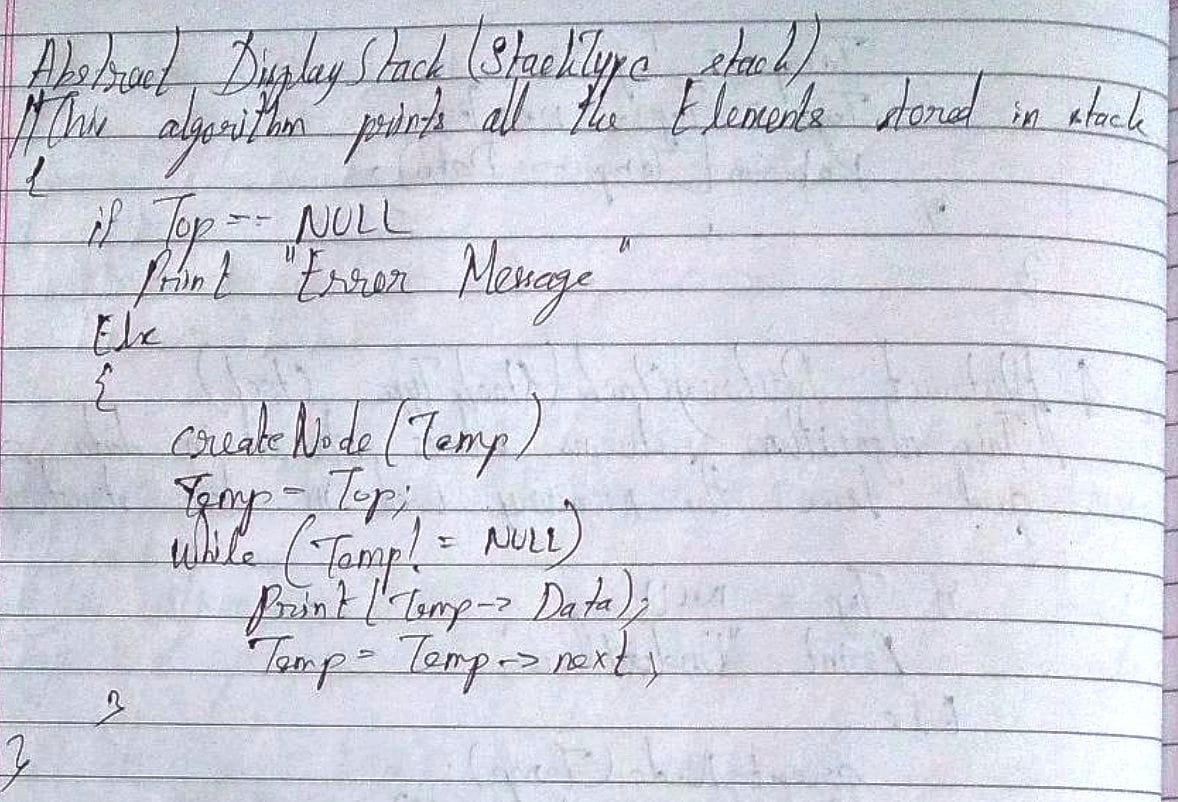
**Linked List Implementation:**

****

****

****

****

****

**Implementation Details:**

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

**Ans.** The steps followed to implement the code are as follows:

* 1. Choice was made – array or linked list implementation
  2. If array implementation was chosen then –
     1. Methods for creating, pushing, popping, peeping and displaying were defined
     2. Main method was created with the necessary code for input and function calling.
     3. Code for Create, Push, Pop, Peep and Display methods was written based on array implementation.
     4. Since it is array implementation in C, the total number of elements in the array has to be taken from the user, maximum in this case being 100.
  3. If linked list implementation was chosen then –
     1. Methods for creating, pushing, popping, peeping and displaying were defined
     2. Structure named “Node” was defined with data field of int data type and address field of struct Node data type
     3. A pointer to the top node was defined and initialized to NULL.
     4. Main method was created with the necessary code for input and function calling.
     5. Code for Create, Push, Pop, Peep and Display methods was written based on array implementation.
     6. Since it is linked list implementation in C, the total number of elements was not needed to be taken from the user.

**Program source code:**

**Array Implementation**

#include<stdio.h>

int st[100], n, e, ch, top;

void create(void);

void push(void);

void pop(void);

void peep(void);

void display(void);

void main()

{

printf("Enter the number of elements in the stack which is less than or equal to 100: ");

scanf("%d", &n);

do

{

printf("\nEnter:\n\t'1' to create (use this option only once in the beginning).\n\t'2' to push an element.\n\t'3' to pop an element.\n\t'4' to peep an element.\n\t'5' to display the stack.\n\t'6' to exit from the program.\n\tEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

create();

break;

case 2:

push();

break;

case 3:

pop();

break;

case 4:

peep();

break;

case 5:

display();

break;

case 6:

printf("\nE\tX\tI\tT\tI\tN\tG\t.\t.\t.");

break;

default:

printf("\nEnter either '1', '2', '3', '4', '5' or '6' only and try again!");

}

}while(ch!=6);

}

void create()

{

top = -1;

printf("\nStack has been created. Now choose another option and don't choose\noption '1' again during the remainder course of the program.");

}

void push()

{

if(top >= n-1)

{

printf("\nStack Overflow!");

}

else

{

printf("\nEnter element to push into stack: ");

scanf("%d", &e);

top++;

st[top] = e;

printf("\nPush successful.");

}

}

void pop()

{

if(top == -1)

{

printf("\nStack Underflow!");

}

else

{

printf("\nThe element popped is %d.", st[top]);

top--;

}

}

void peep()

{

if(top == -1)

{

printf("\nStack Underflow!");

}

else

{

printf("\nThe element at top position of stack is %d.", st[top]);

}

}

void display()

{

if(top == -1)

{

printf("\nStack Underflow!");

}

else

{

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

{

printf("\nThe element at position %d is %d.", (i+1), st[i]);

}

}

}

**Linked List Implementation**

#include<stdio.h>

#include<stdlib.h>

int e, ch;

void create(void);

void push(void);

void pop(void);

void peep(void);

void display(void);

struct Node

{

int data;

struct Node \*next;

};

struct Node \*top = NULL;

/\*The stack is created in this line utself.

The create function is added just as a formality.

If the statement at line 14 is placed inside

the create function, then other methods cannot

access it.\*/

void main()

{

printf("\nAs this is Linked List implementation, there is no need to enter the total number of elements.");

do

{

printf("\nEnter:\n\t'1' to create (use this option only once in the beginning).\n\t'2' to push an element.\n\t'3' to pop an element.\n\t'4' to peep an element.\n\t'5' to display the stack.\n\t'6' to exit from the program.\n\tEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

create();

break;

case 2:

push();

break;

case 3:

pop();

break;

case 4:

peep();

break;

case 5:

display();

break;

case 6:

printf("\nE\tX\tI\tT\tI\tN\tG\t.\t.\t.");

break;

default:

printf("\nEnter either '1', '2', '3', '4', '5' or '6' only and try again!");

}

}while(ch!=6);

}

void create()

{

printf("\nStack has been created. Now choose another option and don't choose\noption '1' again during the remainder course of the program.");

}

void push()

{

struct Node \*temp;

temp = (struct Node\*)malloc(sizeof(struct Node));

printf("\nEnter the element to push: ");

scanf("%d", &temp -> data);

temp -> next = NULL;

if(top == NULL)

{

top = temp;

}

else

{

temp -> next = top;

top = temp;

}

printf("\nPush successful.");

}

void pop()

{

struct Node \*temp;

if(top == NULL)

{

printf("\nStack Underflow!");

}

else

{

temp = top;

printf("\nThe element popped is %d.", temp -> data);

top = top -> next;

free(temp);

printf("\nPop successful.");

}

}

void peep()

{

if(top == NULL)

{

printf("\nStack Underflow!");

}

else

{

printf("\nThe element at top position is %d.", top -> data);

}

}

void display()

{

struct Node \*temp;

if(top == NULL)

{

printf("\nStack Underflow!");

}

else

{

int i = 0;

temp = top;

while(temp!=NULL)

{

printf("\nThe element at position %d is %d.", (i+1), temp -> data);

temp = temp -> next;

i++;

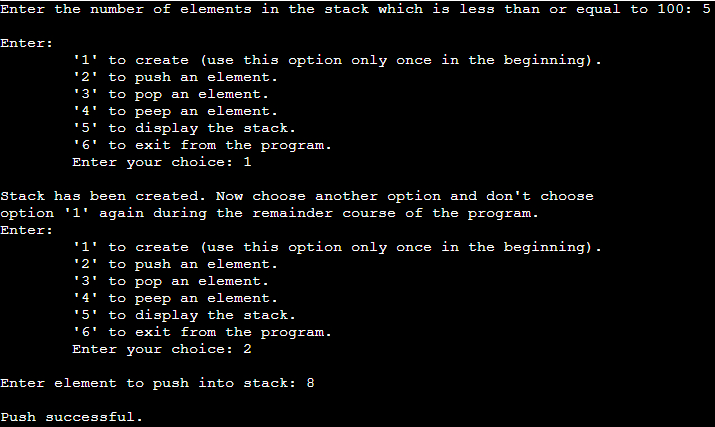
}

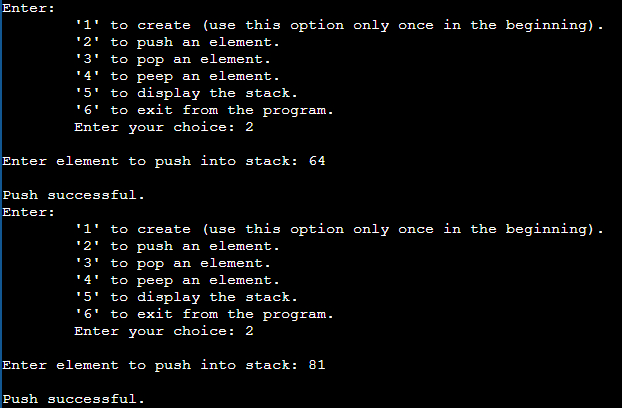
}

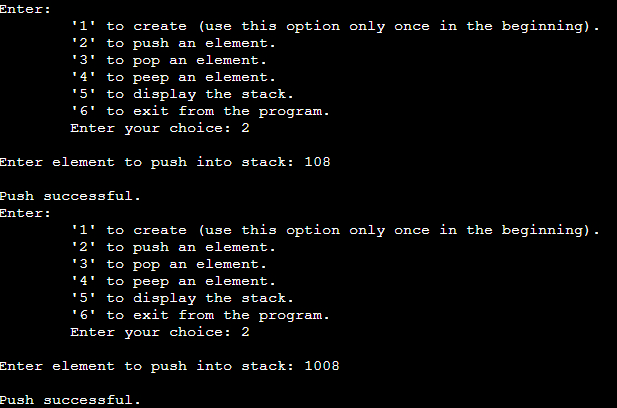
}

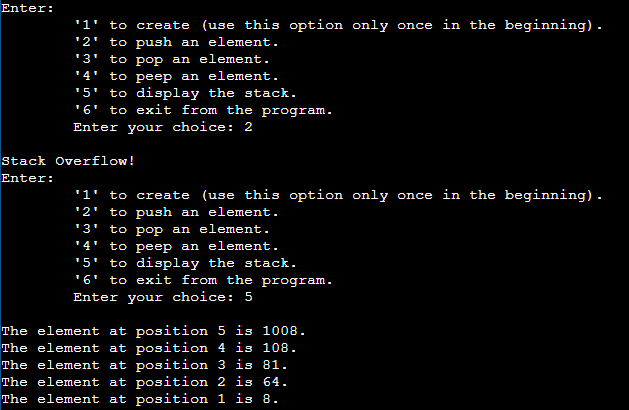
**Output Screenshots:**

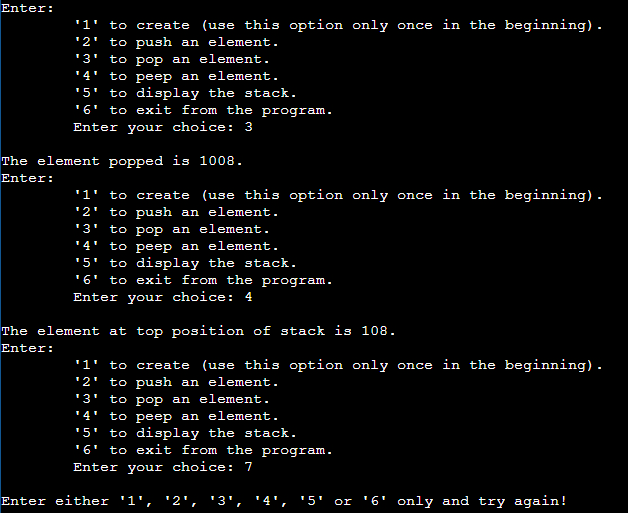
**Array Implementation:**

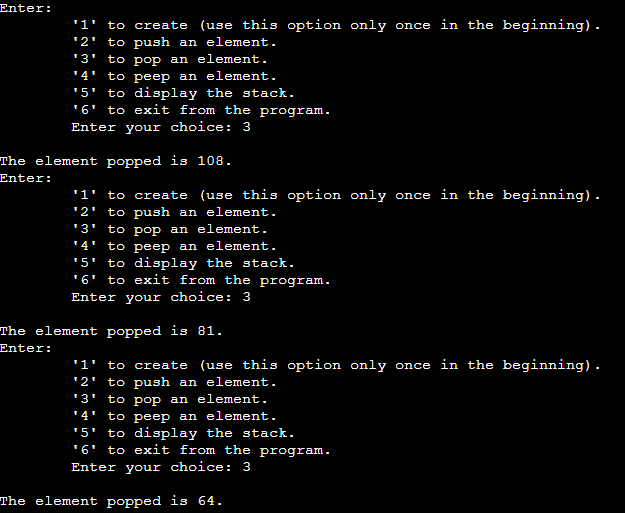
****

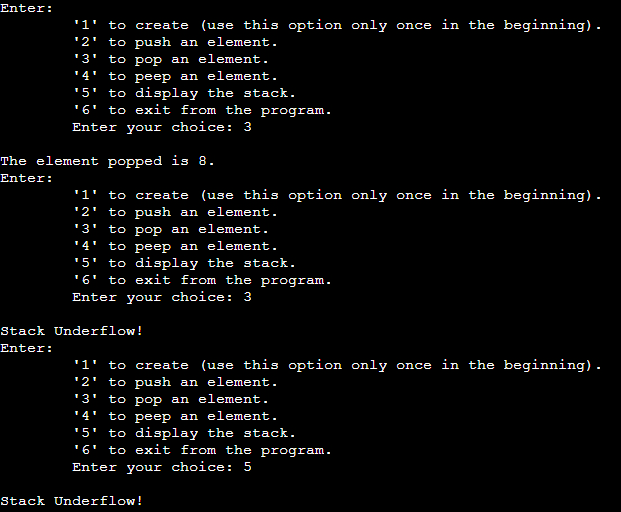
****

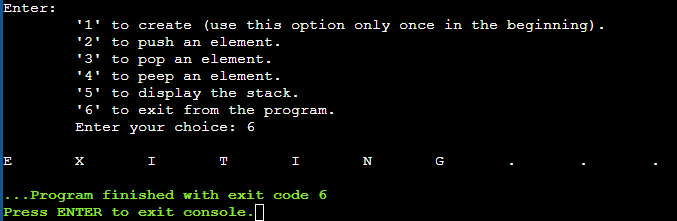
****

****

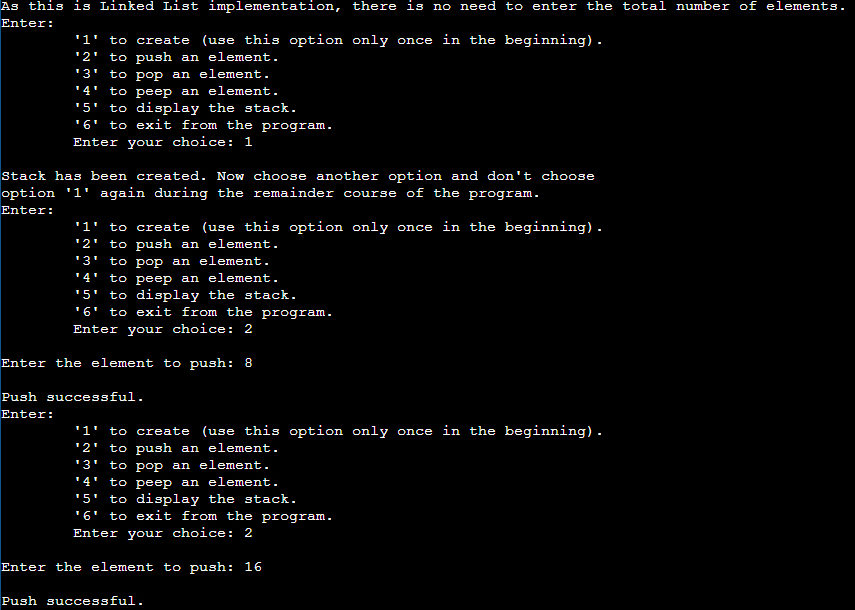
****

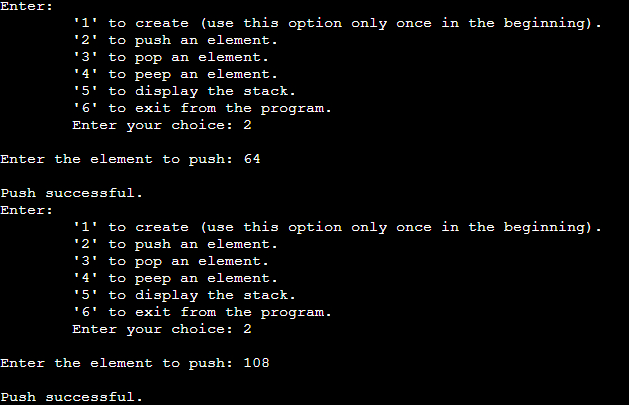
****

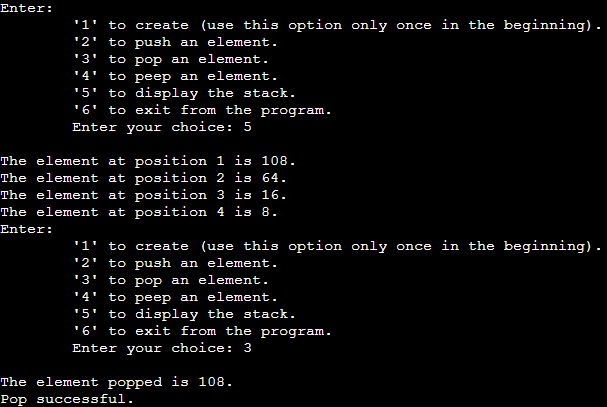
****

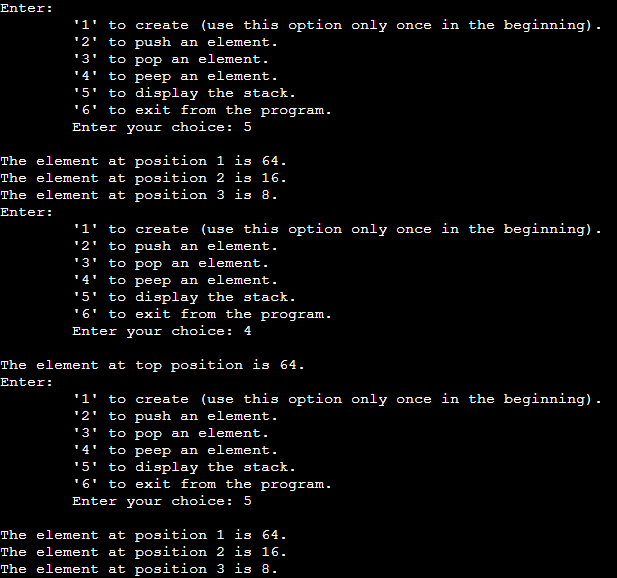
****

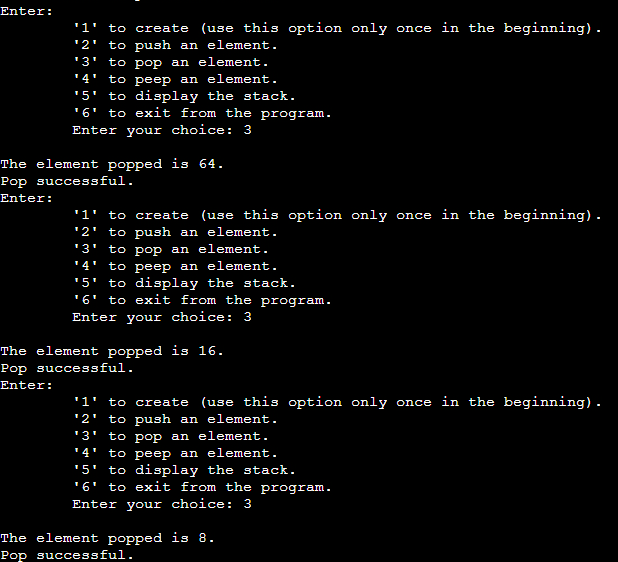
**Linked List Implementation:**

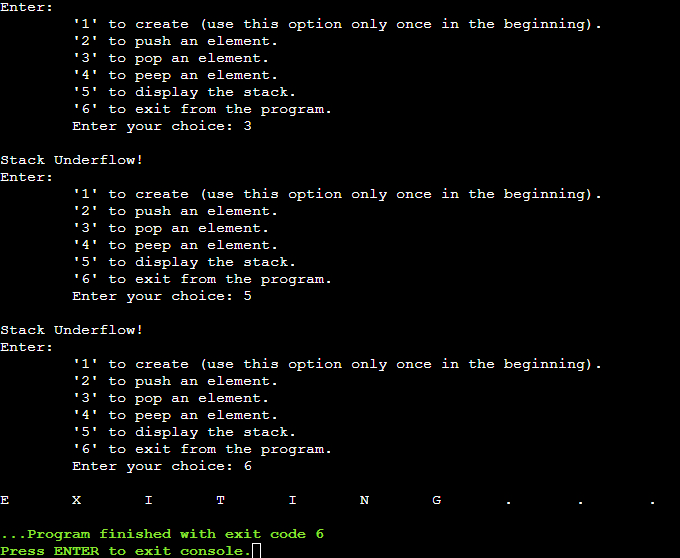
****

****

****

****

****

****

**Applications of Stack:**

* + - 1. To store history in a web browser
      2. Undo operation in any application or text editor
      3. Saving local variables during function calls
      4. To test whether a string or a number is palindrome
      5. Parenthesis Matching Algorithm is based on the principle of stack and is used by compilers in IDE’s to check if the code has got proper placement of brackets.

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

Ans. Array implementation is important as it is easy to access an element from any position in an array (by specifying the array index number). Linked List implementation is important as it is easy to add or remove an element (by changing the pointers). Both approaches are dynamic in nature, but Linked List is even more so because it doesn’t even ask the user for the maximum number of nodes, unlike an array. So Linked List implementation is fully dynamic and hence its size can be increased or shrunk as per requirement, which is the case in most real-life applications. So stacks can be best implemented using Linked List implementation. Stacks find use in various computer applications.

**Conclusion:-**

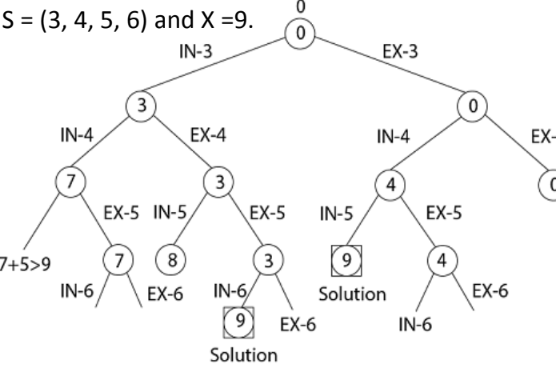
Thus, in this experiment, the implementation of a stack using Array and Linked List has been shown. The operations which include Create, Push, Pop, Peep and Display have been included.

**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 given time. Stacks are used to store the solution path.

For example,

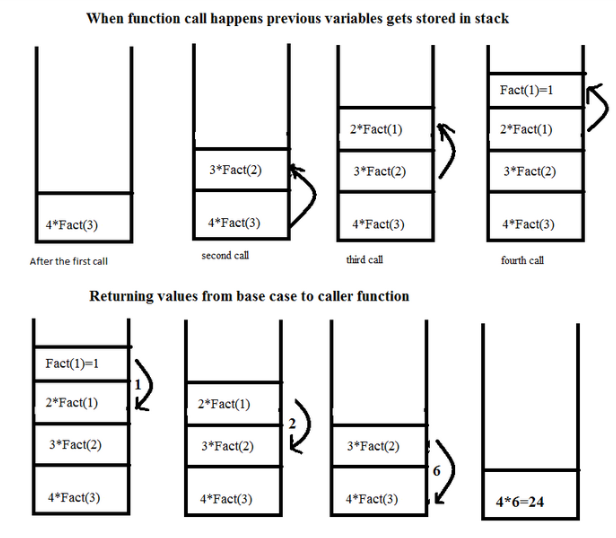


So the Stack can store the solution path as: 0 -> IN-3 -> EX-4 -> EX-5 -> IN-6

0 -> EX-3 -> IN-4 -> IN-5

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

**Ans.** During a Recursion, the current function is paused. The current values of parameters, local variables and the return address (the address where the control has to return after the function call is completed) are stored in a stack called the Call Stack. First the arguments, then the return address and then the local variables are pushed onto the stack. Once the recursive function call is completed, the contents of the Call stack are loaded back and the control finds the address from where it has to continue the execution of the program and the program execution continues.

****

In the above example, the factorial of 4 is found using recursion. After the first call, the result contains 4 multiplied by another function call with 3 passed as parameter. After the second call, the result stack contains 3 multiplied by another function call with 2 passed as parameter, in addition to the previous contents. After the third call, the result stack contains 2 multiplied by another function call with 1 passed as parameter, in addition to the previous contents. After the fourth call, the result stack contains 1 after which there are no more calls, in addition to the previous contents. Then, the stack elements are deleted in LIFO order such that every time before a top element is deleted, it is first applied with the element next to top. This goes on until there is one last element remaining in the stack, which is the result.