Batch: B2 Roll No.: 16010122151
Experiment / assignment / tutorial No. 3
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

2

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*

NAME: SMIT SANTOSH PATIL

ROLL NO.: 16010122139

BATCH: B2

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: -

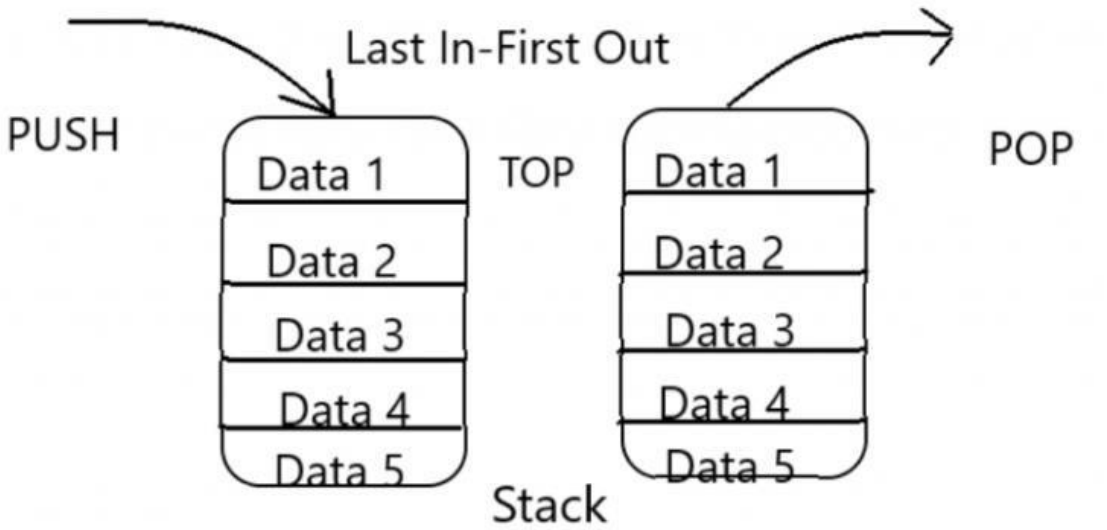
Stack is a linear data structure which follows a particular order in which the operations are performed. It works on the mechanism of Last in First out (LIFO).

List 5 Real Life Examples:

- **A book pile**
- **Call Stack in Programming**
- **Undo/Redo functionality in Softwares**
- **Browser Back Button**
- **Plates in a Cafeteria**
- **Stacking chairs**

Diagram:

Operations on stack



Explain Stack ADT:

Push: Adds an element to the top of the stack.

Pop: Removes and returns the top element from the stack.

Peek (or Top): Returns the top element of the stack without removing it.

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

1. Stack Creation: The stack is initialized with an empty state by setting top to -1.

2. Push (Insertion) Operation: This operation adds an element to the top of the stack.

3. Pop (Deletion) Operation: This operation removes and returns the top element from the stack.

4. Display Operation: This operation displays all elements currently in the stack.

Implementation Details:

Assumptions made for Input:

Global variables :

stack[100]: This is the integer array used to store stack elements.

choice: It stores the user's choice for the operation to be performed

n: The maximum size of the stack (50 in this case).

top: It represents the top of the stack. It's initialized to -1, indicating that the stack is initially empty.

x: A variable to store the value to be pushed onto the stack.

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

- Include<stdio.h>
- Main()
- Push()
- Pop()
- Display()

Program source code:

Code for static stack :

```
#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);
```

```
void display(void);

int main()

{

   top=-1;

   n=50;

   printf("\n\t STACK OPERATIONS USING ARRAY");

   printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY ALL THE ELEMENTS\n\t
4.EXIT");

   do

   {

      printf("\n Enter the Choice:");

      scanf("%d",&choice);

      switch(choice)

      {

         case 1:

         {

            push();

            break;

         }

         case 2:

         {

            pop();

            break;

         }

         case 3:
```

```
        {
           display();
           break;
        }
        case 4:
        {
           printf("\n\t EXIT ");
           break;
        }
        default:
        {
           printf ("\n\t Please enter a Valid Option");
        }

     }
   }
   while(choice!=4);
   return 0;
}
void push()
{
   if(top>=n-1)
   {
      printf("\n\tSTACK is over flow");
```

```
    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
```

```
        printf("\n The elements in STACK:");
        for(i=top; i>=0; i--)
            printf("\n %d",stack[i]);
    }
    else
    {
        printf("\n The STACK is empty");
    }
}
```

Output Screenshots:

```
STACK OPERATIONS USING ARRAY
     1.PUSH
     2.POP
     3.DISPLAY ALL THE ELEMENTS
     4.EXIT
 Enter the Choice:1
 Enter a value to be pushed:25
 Enter the Choice:1
 Enter a value to be pushed:28
 Enter the Choice:1
 Enter a value to be pushed:30
 Enter the Choice:2
 The popped elements is 30
 Enter the Choice:3
 The elements in STACK:
 28
 25
 Enter the Choice:4
 EXIT
```

CODE FOR DYNAMIC IMPLEMENTATION OF STACK:

```
#include <stdio.h>

#include <stdlib.h>


struct node {

   struct node *next;

   int data;
```

```
};

void push(int value, struct node ** top)
{
   struct node * n = malloc(sizeof(struct node));
   n->data = value;
   n->next = NULL;

   if(top == NULL)
   {
      *top = n;
   }
   else
   {
      n->next = *top;
      *top = n;
   }
}

void peek (struct node * top)
{
   printf("%d is the number at the top\n", top->data);
}

void pop(struct node ** top)
{
   if(top == NULL)
```

```
    {
        printf("The stack is empty\n");
        return;
    }
    struct node *temp = *top;
    *top = (*top)->next;
    free(temp);
    return;
}

int main()
{
    struct node* top = NULL;
    push(5, &top);
    peek(top);
    push(7, &top);
    peek(top);
    pop(&top);
    peek(top);
}
```

```
5 is the number at the top
7 is the number at the top
5 is the number at the top
```

Applications of Stack:

- Expression evaluation and syntax parsing
- Balancing of symbols, paranthesis
- Infix to Postfix /Prefix conversion
- Reverse a String using Stack
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Backtracking
- Compile time memory management

Explain the Importance of the approach followed by you

The code written above is a simple implementation of a stack using an array and offers a menu-driven interface to perform basic stack operations like push, pop, and display.

- Understanding easier: This code provides a practical example of implementing a fundamental data structure, the stack.
- Menu-Driven Interface: The menu-driven interface enhances user experience and interaction with the stack
- Error Handling: The code includes error checking for stack overflow and underflow conditions.
- Practical Relevance: Stacks have numerous real-world applications, such as function call management, undo/redo functionality in software, parsing expressions, and more

- Code Readability: The code is relatively simple and well-commented, making it accessible to learners at different levels of programming experience.

Conclusion:-

Thus , we learned implementation of Basic operations on stack using Array and Linked List.

PostLab Questions:

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

1) 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

Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver

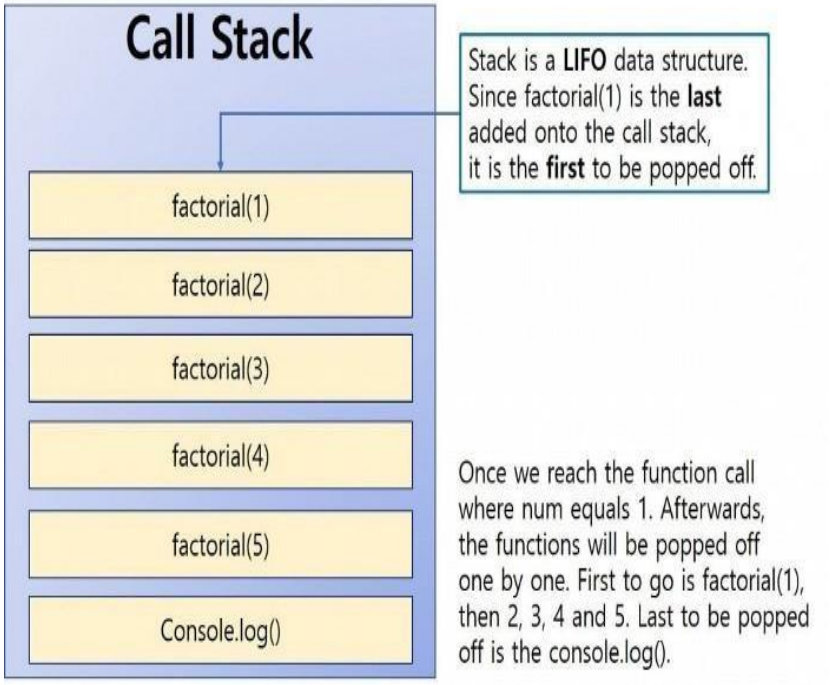
Example: Finding the correct path in a maze.

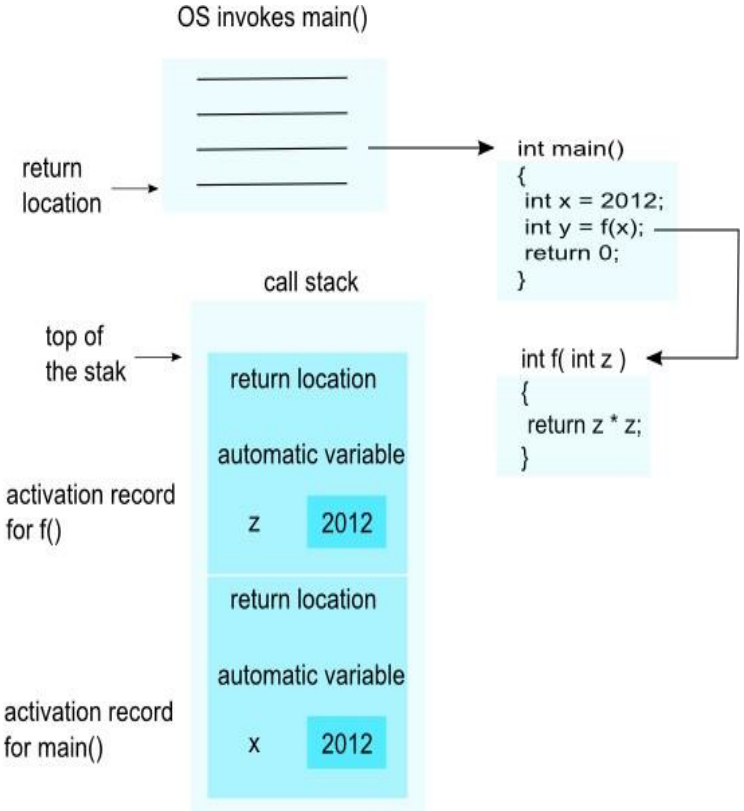
- There are a series of points, from the starting point to the destination.
- We start from one point. To reach the final destination, there are several paths.
- Suppose we choose a random path. After following a certain path, we realise that the path we have chosen is wrong. So we need to find a way 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 top point from the stack and thus return to the last point and continue our quest to find the right path.

2) Stacks are an important way of supporting nested or recursive function calls.

- The ways in which subroutines receive their parameters and return results—use a special stack (the "call stack")
- Call stack holds information ,about procedure/function calling and nesting , in order to switch to the context of the called function and restore to the caller function when the calling finishes.


Call Stack
Stack is a LIFO data structure. Since factorial(1) is the last added onto the call stack, it is the first to be popped off.
factorial(1)
factorial(2)
factorial(3)
factorial(4)
factorial(5)
Console.log()
Once we reach the function call where num equals 1. Afterwards, the functions will be popped off one by one. First to go is factorial(1), then 2, 3, 4 and 5. Last to be popped off is the console.log().


OS invokes main()
return location
int main()
{
int x = 2012;
int y = f(x);
return 0;
}
call stack
top of the stak
int f(int z)
{
return z * z;
}
return location
automatic variable
activation record for f()
z
2012
return location
automatic variable
activation record for main()
x
2012