Batch: D3	**Roll No.: 16010123294**
Experiment / assignment / tutorial No. 2	
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 - 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.

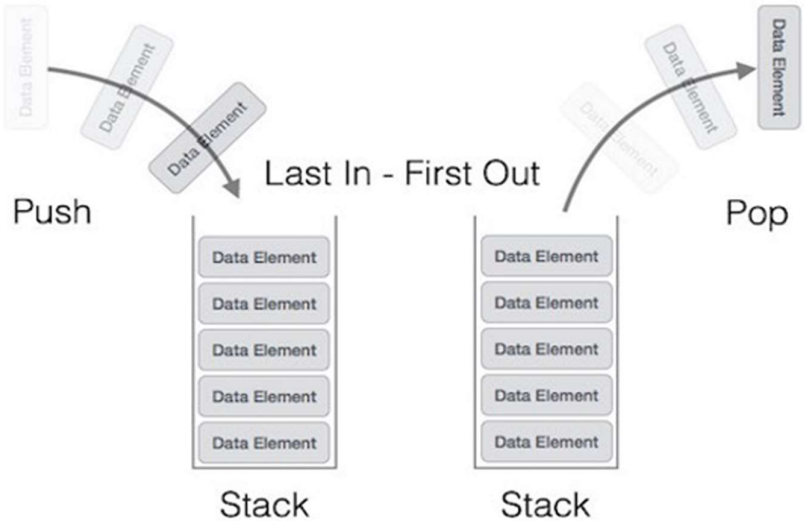
Related Theory: -

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 where we use stack:

a) Browsing history
b) undo/redo functions
c) function calling
d) expression evaluation
e) Backtracking algorithms

Diagram:



Explain Stack ADT:

A Stack Abstract Data Type (ADT) is a linear data structure that operates on the Last In, First Out (LIFO) principle, where the last element added is the first to be removed. Key operations include push (adding an element), pop (removing the top element), peek (viewing the top element without removal), isEmpty (checking if the stack is empty), and isFull (for fixed-size stacks, checking if the stack is full). Stacks are widely used in managing function calls, expression evaluation, and implementing undo mechanisms in software. They can be implemented using arrays (fixed size) or linked lists (dynamic size), akin to a stack of plates where only the top plate can be added or removed.

Algorithm for creation, insertion, deletion, displaying an element in stack [static implementation]:

Initialize Stack:

Define MAX as 10.

Declare an array arr[MAX] to store stack elements.

Set top to -1 to indicate an empty stack.

Check if Stack is Full:

Function isFull() returns true if top is equal to MAX-1, indicating the stack is full.

Check if Stack is Empty:

Function isEmpty() returns true if top is equal to -1, indicating the stack is empty.

Push Operation:

Function push(int num):

If the stack is full, print "Stack is Full".

Otherwise, increment top by 1 and assign num to arr[top].

Print the pushed value.

Pop Operation:

Function pop():

If the stack is empty, print "Stack Underflow".

Otherwise, print the top element, then decrement top by 1.

Display Stack:

Function display():

If the stack is empty, print "Stack is empty".

Otherwise, print all elements from arr[top] to arr[0].

Peek Operation:

Function peek():

If the stack is empty, print "Stack is empty".

Otherwise, print the top element.

Main Function:

Loop infinitely to present a menu to the user:

Push

Pop

Peek

Display

Exit

Read the user's choice.

Based on the choice, call the corresponding function (push, pop, peek, display).

For choice 5, call exit(0) to terminate the program.

If an invalid choice is entered, print an error message.

Implementation Details:

Assumptions made for Input:

User will enter Integer value only while pushing the elements .

Built-In Functions/Header Files Used:

#include<stdio.h>

#include<stdlib.h>

exit();

Program source code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int arr[MAX];
int top=-1;

int isFull(){
return top == MAX -1;
}

int isEmpty(){
return top ==-1;
}
void push(int num){
```

```
if(isFull())
   printf("Stack is Full\n");
else{
                top=top+1;
                arr[top] = num;
                printf("%d is pushed to Stack",num);

}

}
void pop(){
if (isEmpty())
                printf("Stack Underflow\n");
        else
        {
                printf("Popped element is : %d\n",arr[top]);
                top=top-1;
        }
}

void display(){

        int i;
        if (isEmpty())
                printf("Stack is empty\n");
        else
```

```
        {
                printf("Stack elements :\n");
                for(i = top; i >=0; i--)
                        printf("%d\n", arr[i] );
        }
}

void peek(){
   if (isEmpty())
      printf("Stack is empty\n");
   else
      printf("Top element is %d",arr[top]);
}

int main() {
   int choice, num;

   while (1) {
      printf("\n Stack Operations:\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Peek\n");
      printf("4. Display\n");
      printf("5. Exit\n");
      printf("Enter your choice(1-5): ");
      scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &num);
                push(num);
                break;
            case 2:
                pop();
                break;
            case 3:
                peek();
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exit the Program\n");
                exit(0);
            default:
                printf("Invalid choice! Please enter a number between 1 and 5.\n");
        }
    }

    return 0;
}
```

Output Screenshots:

PUSH ()

```
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 10
10 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 20
20 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 30
30 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 40
40 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 50
50 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
```

```
50 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 60
60 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 70
70 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 80
80 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 90
90 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 100
100 is pushed to Stack
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
```

POP()

```
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 4
Stack elements :
100
90
80
70
60
50
40
30
20
10

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 100

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 4
Stack elements :
90
80
70
60
50
40
30
20
10

 Stack Operations:
1. Push
2. Pop
```

```
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 3
Top element is 90
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 90

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 80

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 70

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 60

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
```

```
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 40

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 30

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 20

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Popped element is : 10

 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Stack Underflow
```

DISPLAY()

```
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 4
Stack elements :
100
90
80
70
60
50
40
30
20
10
```

Stack Overflow

```
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 1
Enter value to push: 110
Stack is Full

 Stack Operations:
1. Push
```

Stack UnderFlow

```
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 2
Stack Underflow
```

EXIT()

```
 Stack Operations:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice(1-5): 5
Exit the Program

Process returned 0 (0x0)   execution time : 134.702 s
Press any key to continue.
```

PostLab Questions:

2) **List 5 Applications of Stack Data Structures.**
   a) Browsing history
   b) undo/redo functions
   c) function calling
   d) expression evaluation
   e) Backtracking algorithms
3) **Convert the given Infix Expression into Postfix Expression using Stack:**

((A-B/C)*(D*E-F))

Scan	Stack	Postfix
(	(	
(	((	
A	((	A
-	((-	A
B	((-	AB
/	((-/	AB
C	((-/	ABC
)	((-	ABC/-
*	(*	ABC/-
(	(*(	ABC/-
D	(*(	ABC/-D
*	(*(*	ABC/-D
E	(*(*	ABC/-DE
-	(*(-	ABC/-DE*
F	(*(-	ABC/-DE*F

)	(*	ABC/-DE*F-
)	NULL	ABC/-DE*F-*

The postfix expression is ABC/-DE*F-*

4) **Explain How stack can be used in both Nested Function calls and Recursion using suitable examples for each. Further Define Activation Records used for Function Calling.**

Stack in nested function call:

Function Arguments: When a function is called, its arguments are pushed onto the stack. These arguments are accessible by the function's code.

Local Variables: Local variables specific to the function are also stored on the stack. This memory is ephemeral and is deallocated when the function returns.

Return Address: The location from which the program should resume execution after the function concludes is stored on the stack as the return address.

Function Body Execution: The function's code is then executed, and the function might perform operations, make calculations, or recursively call other functions.

Function Return: Upon reaching the function's return statement, or upon encountering the end of the function's code block, the function returns. At this point, the activation record pertaining to the function call is popped from the stack. The return value, if any, is passed back to the calling function.

Stack in recursion:

Recursive Call: During a recursive function call, a new activation record is generated on the stack for the recursive call. This record stores the arguments specific to the recursive call, along with the local variables and the return address.

Nested Calls: As the recursion continues, more activation records are pushed onto the stack, one for each recursive call. This creates a stack frame, essentially a stack of function calls.

Base Case Reached: When the recursive function encounters its base case, the condition that terminates the recursion, the function starts returning.

Stack Unwinding: As the function returns from the base case, the corresponding activation record is popped from the stack. The function's local variables and any return value are dealt with.

Function Return: The function then returns to the point where it was called from (the caller), and the return value is processed. This process continues as each function call in the recursion stack unwinds.

Conclusion:-

Learned about stack data structure and its implementation in different daily tasks and also used it in the code to show various operations of stack.