K. J. Somaiya School of Engineering, Mumbai-77
(Somaiya Vidyavihar University)

Batch: B2	**Roll No.: 16010124107**
Experiment / assignment / tutorial No. 1	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

t

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:

1. Placing books on a bookshelf
2. Adding dirty laundry in the hamper
3. Packing suitcase before vacation
4. Entering local train during rush hour
5. Nerf gun

Diagram:

Explain Stack ADT:

A stack is an Abstract Data Type (ADT) that represents a collection of elements with a LIFO principle: Last In, First Out. This means the last element added to the stack is the first one to be removed.

Abstract typedef StackType(ElementType ele) this implies creating an abstract data type called StackType which works on a generic ElementType called ele. A real time implementation of this line could be:

typedef stack(int x)

For operator definitions, we have Abstract StackType CreateEmptyStack() for example, which creates an empty stack. Abstract StackType PushStack(StackType stack, ElemetnType ele) takes element ele and pushes it into stack. But before it does so, it checks if the stack is full and if it is, it returns an error message. Similarly, Abstract StackType PopStack(StackType stack) works. It checks if the stack is empty, returns error if yes, else it pops the topmost element from the stack since the stack works on LIFO principle.

Abstract DestroyStack(StackType Stack) works as a deletor for the entire stack. It pops elements repeatedly till the top=-1, and hence destroys the stack completely. Abstract Boolean NotFull(StackType stack) will check if there's more room in the stack. Abstract Boolean NotEmpty(StackType stack) is just the opposite: it checks if the stack is empty. Abstract ElementType Peep(StackType stack) works like a non-deletion pop function. It prints the topmost element and does not remove anything.

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

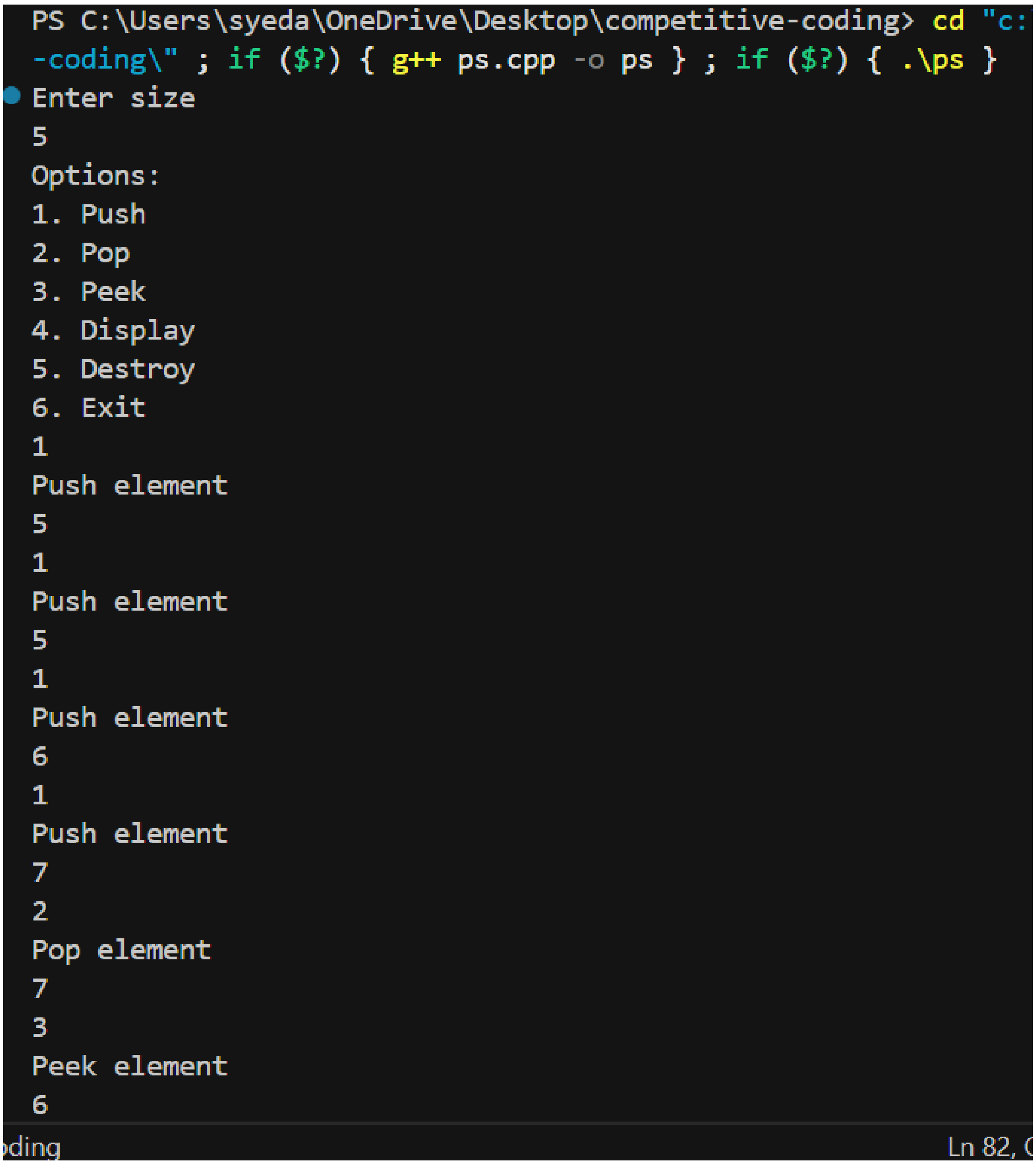
1. start
2. input size of stack from user
3. initialise top as -1
4. initialise option as 0 and print a menu option for the user:
   Options: 1. Push 2. Pop 3. Peek 4. Display 5. Destroy 6. Exit
5. Run a loop till the user does not output exit.
6. If choice=1, check if stack is not full and push the element. Update top.
7. If choice=2, check if the stack is not empty and pop the element. Update top.
8. If choice=3, check if the stack is not empty, and print the top element.
9. If choice=4, iterate from top to 0, print all elements and do not update top.
10. If choice=5, iterate from top to 0, print all elements and update top as you go.
11. If choice=6, terminate switch and break out.
12. Print invalid choice if an unexpected input is entered.

13. Stop.

Program source code:

```
#include <bits/stdc++.h>
using namespace std;
#define int long long

signed main()
{
    cout << "Enter size\n";
    int size;
    cin >> size;
    int stack[size];
    int top = -1;
    int op=0;
        cout << "Options: \n1. Push\n2. Pop\n3. Peek\n4. Display\n5. Destroy\n6. Exit\n";
    while(op<=6)
    {
    cin >> op;
    switch(op)
    {
        case 1:
        if(top<size-1)
        {
        cout << "Push element\n";
        cin >> stack[++top];

        }

        else

        {

        cout << "Stack is full\n";

        }

        break;


        case 2:

        if(top>-1)

        {

        cout << "Pop element\n";

        cout << stack[top--] << "\n";

        }

        else

        {

        cout << "Stack underflow\n";

        }

        break;


        case 3:

        if(top>-1)

        {

            cout << "Peek element\n";

            cout << stack[top] << "\n";
        }

        else

        {

            cout << "Stack is empty\n";

        }

        break;


        case 4:

        if(top>-1)

        {

            cout << "Stack elements\n";

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

            {

                cout << stack[i] << "\n";

            }

        }

        else

        {

            cout << "Stack is empty\n";

        }

        break;


        case 5:

        if(top==-1)

        {
            cout << "Stack is empty\n";

        }

        else

        {

            while(top>-1)

            {

                cout << stack[top--] << "\n";

            }

        }

        break;


        case 6:

              cout << "Exit\n";

        return 0;


        default:

        cout << "Invalid choice\n";

        break;

    }

    }

}
```

Output Screenshots:



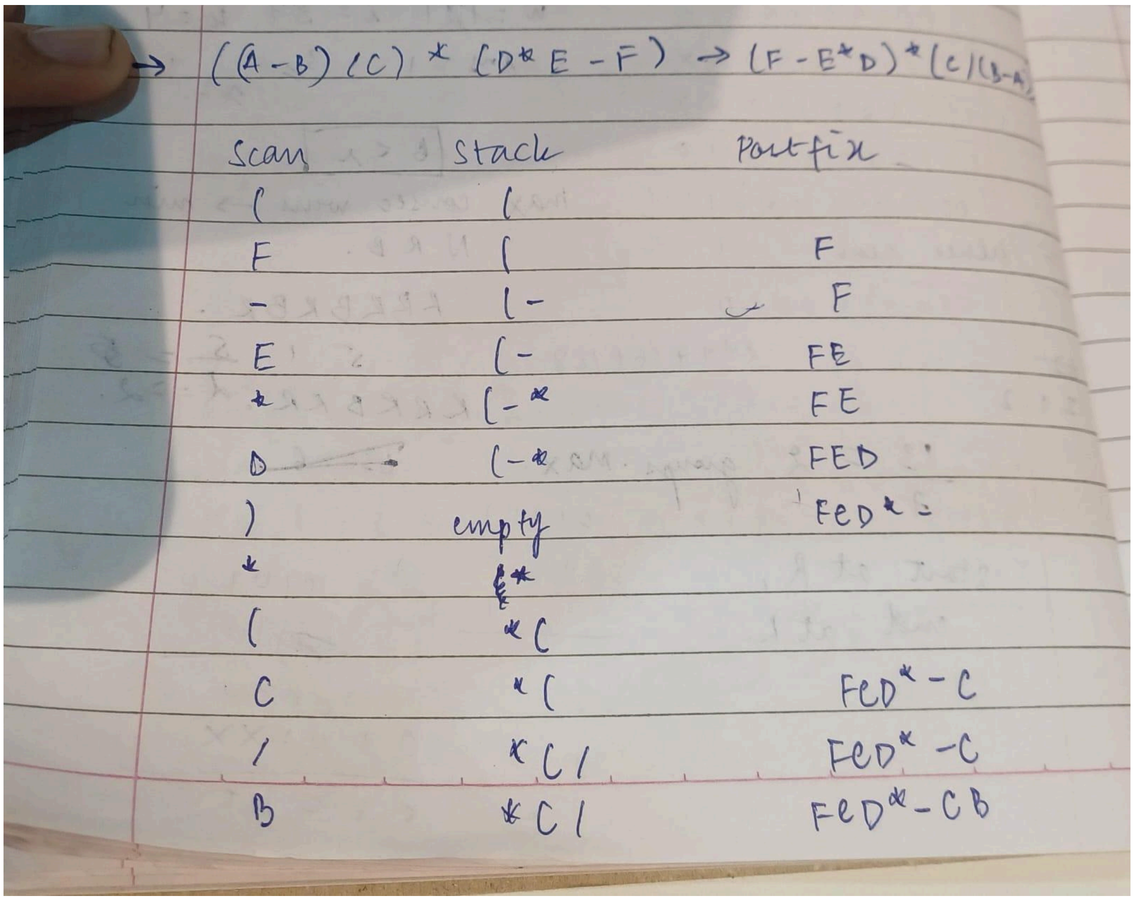
```
PS C:\Users\syeda\OneDrive\Desktop\competitive-coding> cd "c:
-coding\" ; if ($?) { g++ ps.cpp -o ps } ; if ($?) { .\ps }
Enter size
5
Options:
1. Push
2. Pop
3. Peek
4. Display
5. Destroy
6. Exit
1
Push element
5
1
Push element
5
1
Push element
6
1
Push element
7
2
Pop element
7
3
Peek element
6
```

PostLab Questions:

1) **List 5 Applications of Stack Data Structures.**
   Stack as a data structure is used in:
   - Browser history
   - Depth first search in CP
   - Undo-redo functionality
   - Recursion Implementation
   - Conversion of infix to prefix/postfix is also done using stacks.

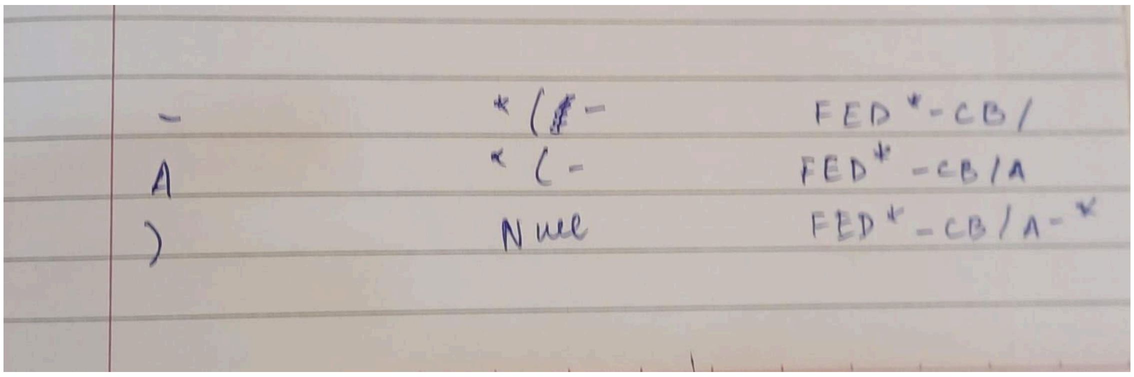
2) **Convert the given Infix Expression into Postfix Expression using Stack:**

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



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

Scan	Stack	Postfix
(	(	
F	(	F
-	(-	F
E	(-	FE
*	(-*	FE
D	(-*	FED
)	empty	FED*-
*	*	
(	*(	
C	*(	FED*-C
/	*(/	FED*-C
B	*(/	FED*-CB



-	*(~~/~~-	FED*-CB/
A	*(-	FED*-CB/A
)	Null	FED*-CB/A-*

3) **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.**

Use of stack in Nested Function call:

If a called function calls another function, a new stack frame for the inner function is created and pushed on top of the existing stack frame. This process continues for any further nested calls. For example, if we call a function A in main, and we call another function B in A. When main() runs, it calls A which further calls B. What happens in the backend is that first, main() is pushed onto the stack during call, then A is pushed as it is called, finally B is pushed on top. When B's functionality is over, it pops and A is on the top, A's functionalities take place, it pops, and finally we go back to main().

Use of stack in recursion:

In factorial or product of first n integer type programs are implemented using recursion, the code looks like this:

```
int prod(int x)

{
        if(x==0) return 1;

        else return x*prod(x-1);

}
```

Let x=5

What happens in the backend?

X is not 0 yet, so the function returns 5*(prod(4))

5 gets stored in the stack, and then prod 4 returns 4*prod3 and so on

When finally X reaches 0, 1 is returned and the stack looks like
5*4 *3 *2* 1* 1 and the final result is output.

Conclusion:-

Stack ADT is a collection of elements that follows the LIFO principle. It is an ordered data type, which means that the order in which the elements are pushed matters. It also allows deletion only from the top. Stack is a powerful tool in problem solving when dealing with DFS and BFS type operations. Backtracking algorithms also involve stack.