**Batch: C3 Roll No.: 16010123217**

**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

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)
5. [*https://www.naukri.com/code360/library/application-of-stack*](https://www.naukri.com/code360/library/application-of-stack)
6. [*https://www.enjoyalgorithms.com/blog/application-of-stack-data-structure-in-programming*](https://www.enjoyalgorithms.com/blog/application-of-stack-data-structure-in-programming)

**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: -**

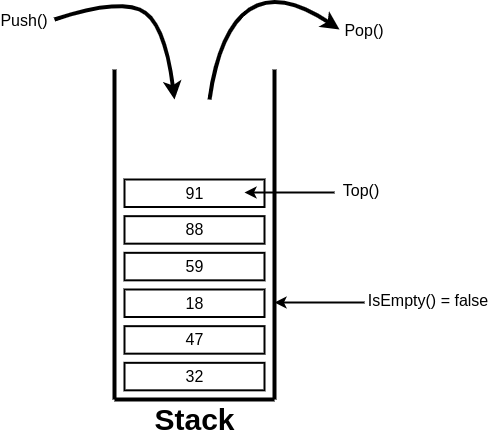
Stack is a linear data structure which follows a particular order in which the operations are performed. It works on the mechanism of Last in First out (LIFO).

**List 5 Real Life Examples where we use stack:**

**Ans:**

1. Back and Forward buttons in a web browser
2. UNDO/REDO functionality in text editors and image editing software
3. Memory management in computer programming
4. Implementing recursion in programming
5. Matching HTML tags in web development

**Diagram:**



**Explain Stack ADT:**

Stack ADT:

Abstract typedef StackType < ElementType ele>

Precondition: none;

Operator definitions:

Abstract StackType CreateStack <ElementType ele>

Precondition: none;

Postcondition: CreateStack is created

Abstract StackType PushToStack <StackType S, ElementType ele>

Precondition: Stack should not be full

Postcondition: PushToStack: Snew  = Sold + ele

Abstract StackType PopStack < StackType S>

Precondition: Stack should not be empty

Postcondition: PopStack: ele = top element in S

Snew  = Sold – ele

Abstract StackType PeekStack < StackType S>

Precondition: Stack should not be empty

Postcondition: PopStack = top element in S

Snew  = original stack

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

**1. Algorithm for creation**

Algorithm StackType CreateStack()

{

integer StackTop =-1;

return stack;

}

**2. Algorithm for insertion**

Algorithm StackType PushStack(StackType Stack, ElementType Element){

if NotFull(Stack)= True

stack[++StackTop]= Element

Else “Error Message”

}

**3. Algorithm for deletion**

Algorithm ElementType PopStack(StackType stack)

[This algorithm accepts a stack as input and returns ‘Element’ at the top of ‘stack’ and deletes that element from stack]

{

if NotEmpty(Stack)= True

return Stack[StackTop--]

else print “Error Message”

}

**4. Algorithm for displaying an element at Top**

Abstract ElementType Peek(StackType stack)

[This algorithm accepts a stack as input and returns Element at the top of the stack. ]

{

if NotEmpty(Stack)= True

return Stack[StackTop]

else print “Error Message”

}

**Implementation Details:**

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

**Program source code:**

#include<bits/stdc++.h>

using namespace std;

*//Declaring the size of stack and top pointing to -1 i.e. stack is empty*

    int top = -1, choice;

    int Stack[10], n = 10;

*//Push*

void push(int val) {

   if(top>=n-1)

   cout<<"Stack Overflow" << endl;

   else {

      top++;

      Stack[top]=val;

      cout<<"Pushed "<<val<<" to the stack\n";

   }

}

*//Pop*

void pop() {

   if(top<=-1)

   cout<<"Stack Underflow"<< endl;

   else {

      cout<<"Popped element: "<< Stack[top] << endl;

      top--;

   }

}

*//Peek*

void peek(){

    cout<<"The top element is: "<< Stack[top] << endl;

}

*//Delete*

void deletestack(){

    if(top == -1){

        cout<<"The stack was already empty\n";

    }

    else{

        top = -1;

        cout<<"Stack deleted successfully\n";

    }

}

*//Display*

void display() {

   if(top>=0) {

      cout<<"Stack elements are: ";

      for(int i=top; i>=0; i--)

      cout<<Stack[i]<<" ";

      cout<<endl;

   } else

   cout<<"Stack is empty\n";

}

int main(){

*//Menu driven program for user*

    while(1){

    cout<<"\nMenu\n";

    cout<<"1. Push\n";

    cout<<"2. Pop\n";

    cout<<"3. Peek\n";

    cout<<"4. Delete Stack\n";

    cout<<"5. Display Stack\n";

    cout<<"6. Exit\n";

    cout<<"Enter your choice:\t";

    cin>>choice;

    switch(choice){

        case 1:

            cout<<"Enter the element to be pushed:\t";

            int ele;

            cin>>ele;

            push(ele);

            break;

        case 2:

            pop();

            break;

        case 3:

            peek();

            break;

        case 4:

            deletestack();

            break;

        case 5:

            display();

            break;

        case 6:

            exit(0);

            break;

        default:

            printf("Invaild choice, please try again\n");

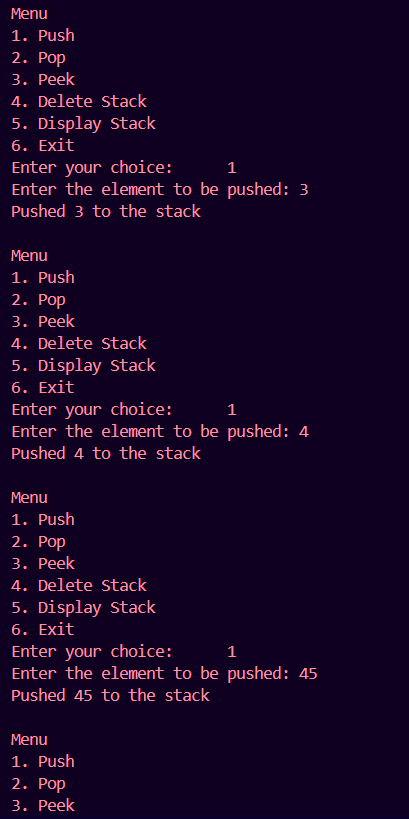
    }

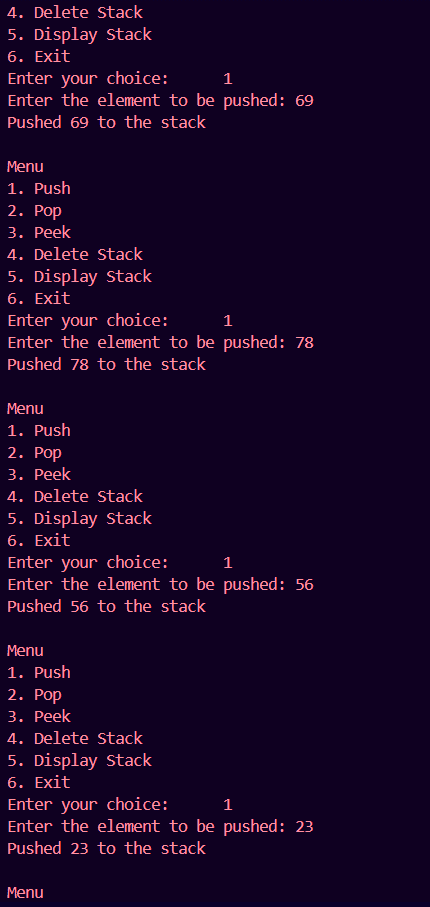
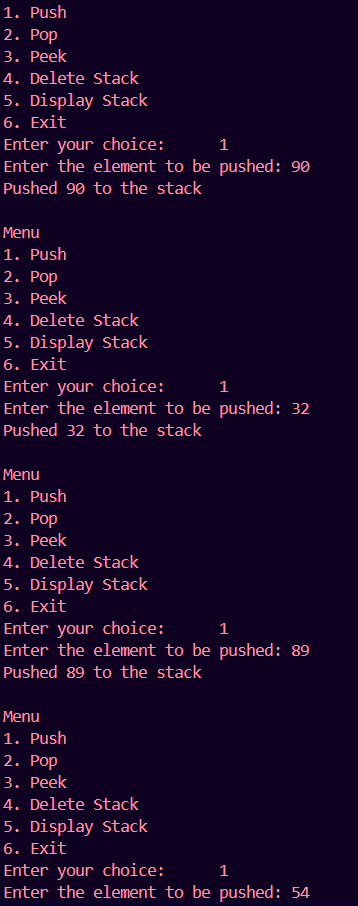
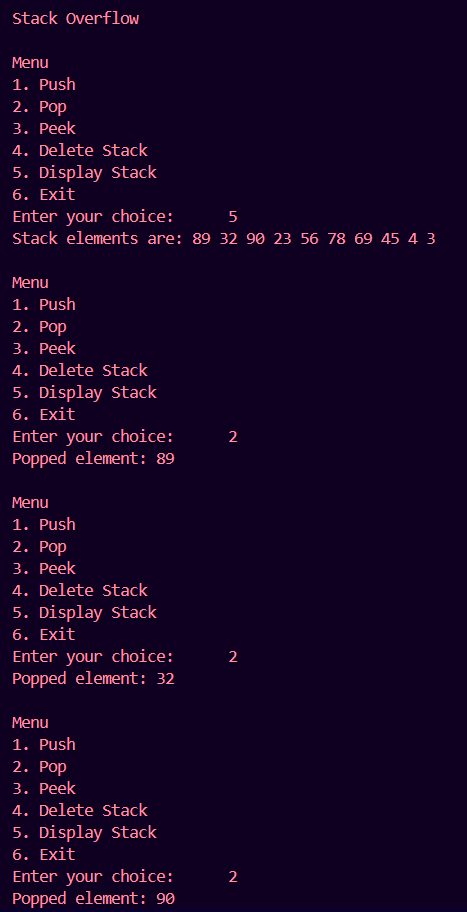
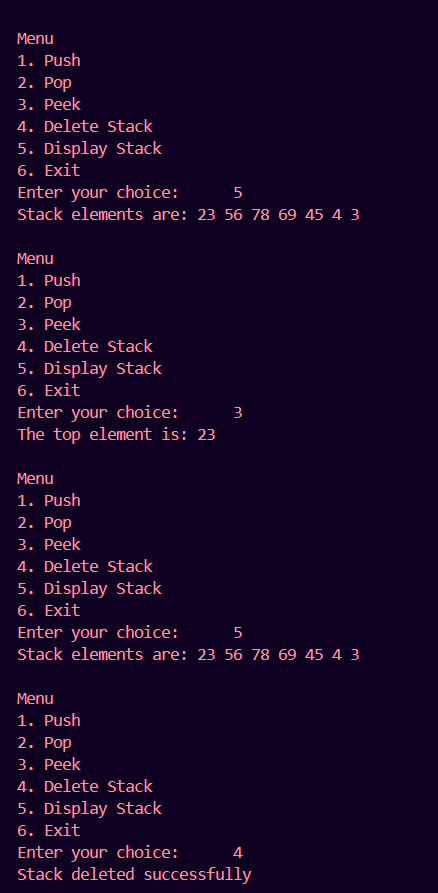
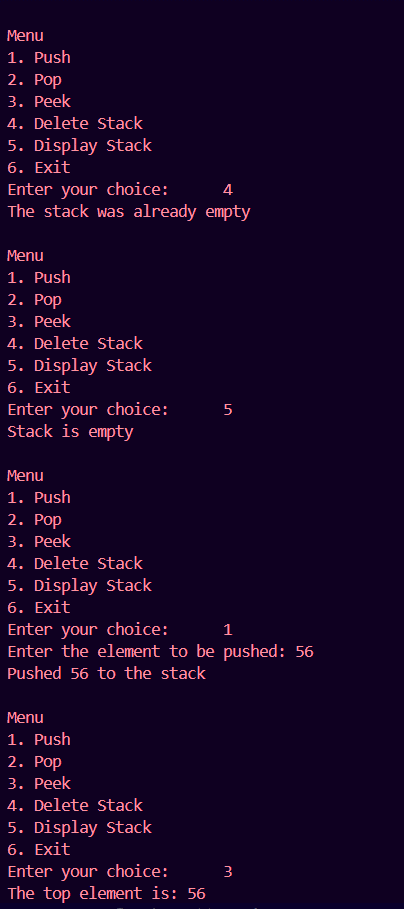
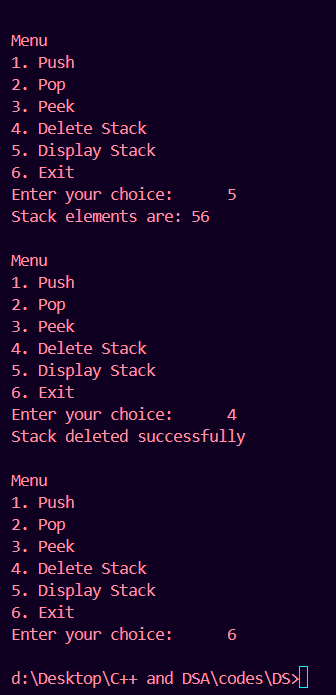
    }

    return 0;

}

**Output Screenshots:**



**Post Lab Questions:**

1. **List 5 Applications of Stack Data Structures.**

**Ans:**

1. Function Calls-

The state of the program is placed into the Stack when a function is invoked. The preceding function's execution is continued after the process returns by popping the state off the Stack.

2. Parenthesis checking-

To determine if brackets are balanced or not, a stack data structure is utilized. An opening parenthesis is popped off the Stack as a closing parenthesis is added onto it. The brackets are balanced if the Stack is empty at the conclusion of the expression.

3. Undo/Redo Operations-

Many apps' undo-redo functionality employs stacks to remember the prior operations. A new action is added to the Stack each time it is completed. The top member of the Stack is popped to undo the action, and the original procedure is then carried out.

4. Web browser history-

Stacks are used by web browsers to record the websites you visit. When you click the back button, the previous URL is removed from the Stack and is added to the Stack each time you visit a new page.

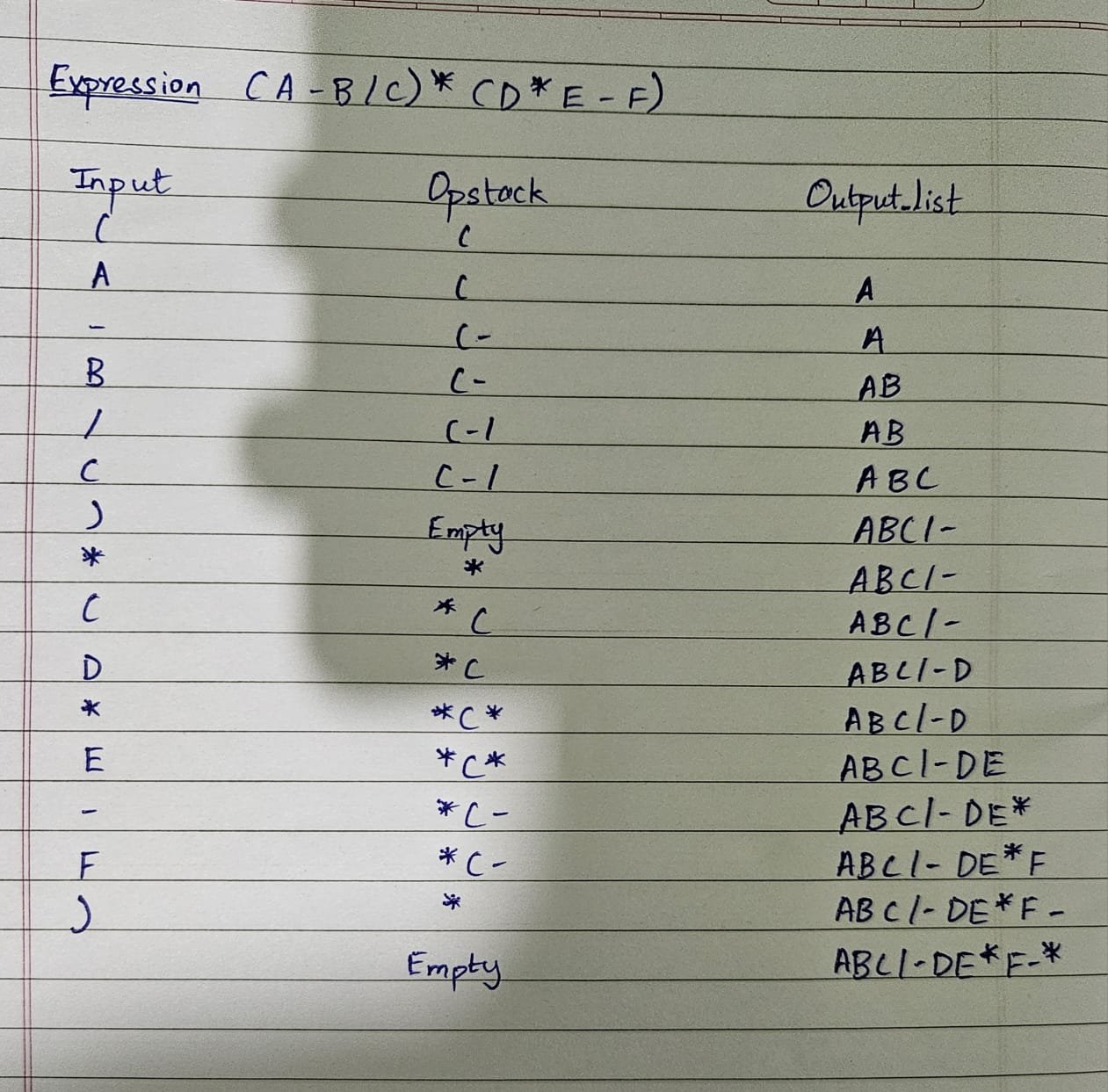
5. Reverse the Data-

We must reorganize the data so that the first and final items are switched; the second and second-last elements are exchanged, and so on for all subsequent elements if we want to reverse a particular collection of data.

1. **Convert the given Infix Expression into Postfix Expression using Stack:**

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

**Ans:**



1. **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.**

**Ans**. Stack Usage in Nested Function Calls and Recursion

1. Nested Function Calls:

In nested function calls, each function call creates an activation record (or stack frame) that stores the function’s return address, local variables, and other data. This record is pushed onto the stack. When a function completes, its stack frame is popped off, and control returns to the caller.

For example:

void functionA() {

functionB(); // Calls functionB

}

void functionB() {

functionC(); // Calls functionC

}

void functionC() {

// functionC execution

}

2. Recursion:

In recursion, a function calls itself, generating a new stack frame for each call. The stack grows with each recursive call and shrinks as each call returns.

For example finding the factorial of a number using recursion

Activation Records:

Activation records (stack frames) store a function's return address, local variables, parameters, and saved registers. They ensure that each function has its own isolated data and that the program can return to the correct point after a function call.

**Conclusion:-**

In this experiment, we implemented stack adt using array. Also we learnt about the various uses of stack in day to day programming. We understood the working behind the creation of stack, insertion, deletion and displaying the element in the stack. Also we understood how to check if the stack is full or empty using the topelement index