K. J. Somaiya College of Engineering, Mumbai
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: D2 Roll No.: 16010122323
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
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:
(Define stack, enlist stack operations).
Stack:

A stack is a linear data structure. Any operation on the stack is performed in LIFO (Last In First Out) order. This means the element to enter the container last would be the first one to leave the container. It is imperative that elements above an element in a stack must be removed first before fetching any element.

Operations on a Stack:

Push:

By pushing, we mean inserting an element at the top of the stack. While using the arrays, we have the index to the top element of the array. So, we'll just insert the new element at the index (top+1) and increase the top by 1. It's intuitive to note that this operation is valid until (top+1) is a valid index and the array has an empty space.

Pop:

Pop means to remove the last element entered in the stack, and that element has the index top. So, this becomes an easy job. We'll just have to decrease the value of the top by 1, and we are done. The popped element can even be used in any way we like.

Peek:

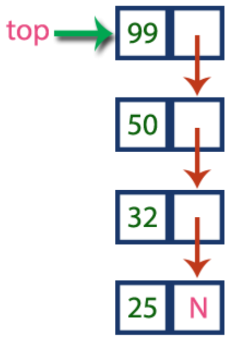
It refers to looking for the element at a specific index in a stack. Peek operation requires the user to give a position to peek at as well. Here, position refers to the distance of the current index from the top element +1

List 5 Real Life applications:

1. Reversing a String
2. Checking for parenthesis matching
3. Infix to Postfix conversion
4. Evaluation of Postfix expression
5. Undo-Redo operations

Algorithm for stack operations using array/Linked list :
To implement a stack using a linked list, we need to set the following things before implementing actual operations.

- **Step 1 -** Include all the **header files** which are used in the program. And declare all the **user defined functions**.
- **Step 2 -** Define a '**Node**' structure with two members **data** and **next**.
- **Step 3 -** Define a **Node** pointer '**top**' and set it to **NULL**.
- **Step 4 -** Implement the **main** method by displaying Menu with list of operations and make suitable function calls in the **main** method.

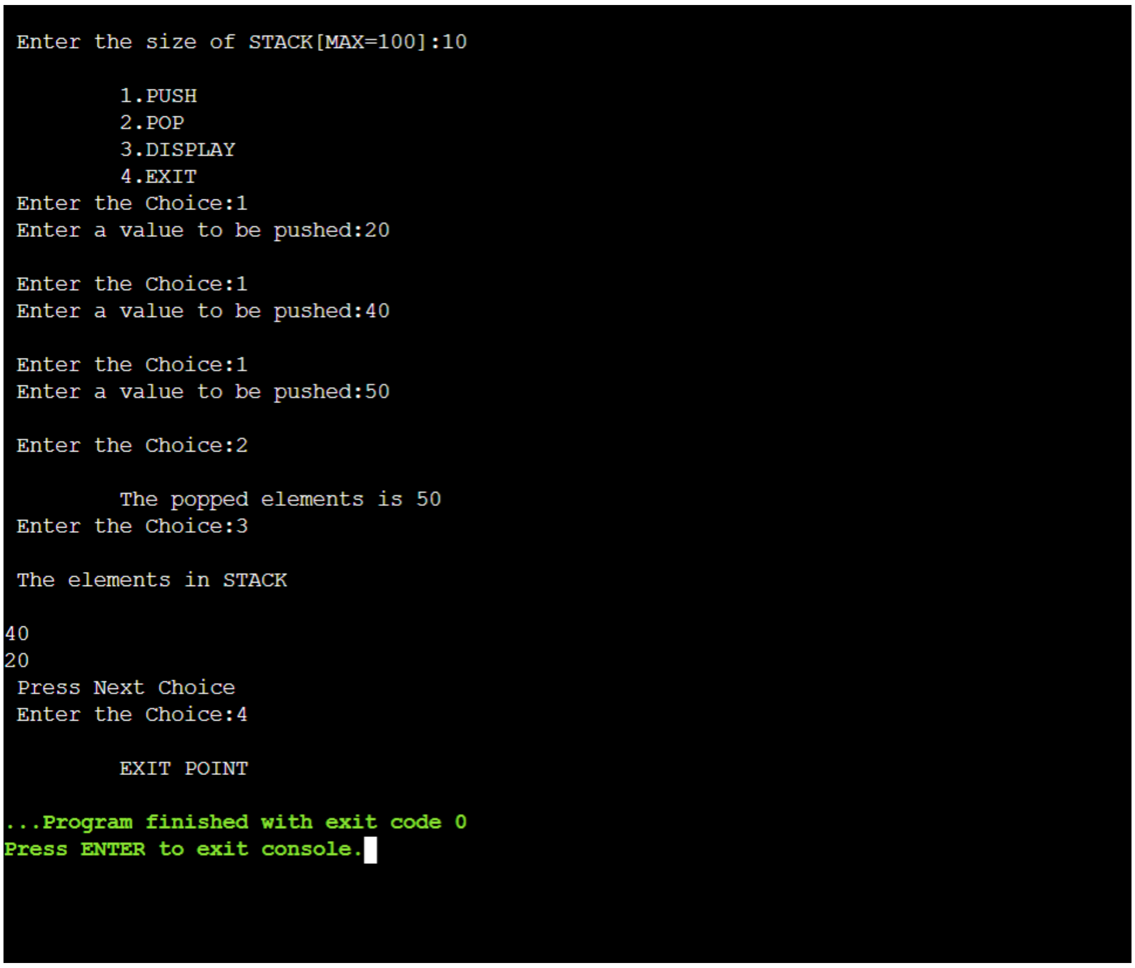


Stack ADT:

Implementation of Stack ADT using Arrays

```
#include <stdio.h>

#define MAX_STACK_SIZE 100

int stack[MAX_STACK_SIZE];
int top = -1;

void push(int value);
int pop();
void display();

int main() {
    int choice;
    printf("\n Enter the size of STACK [MAX=100]: ");
    int n;
    scanf("%d", &n);
    if (n > MAX_STACK_SIZE) {
        printf("Error: Stack size exceeds the maximum limit.\n");
        return 1;
    }

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

    do {
        printf("\n Enter the Choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                {
                    int x;
                    printf("Enter a value to be pushed: ");
                    scanf("%d", &x);
                    push(x);
                    break;
                }
            case 2:
                {
                    int poppedValue = pop();
                    if (poppedValue != -1) {
                        printf("The popped element is %d\n",
poppedValue);
                    }
                    break;
                }
            case 3:
                display();
                break;
            case 4:
                printf("\n EXIT POINT\n");
                break;
            default:
                printf("Please Enter a Valid Choice (1/2/3/4)\n");
        }
    } while (choice != 4);

    return 0;
}

void push(int value) {
    if (top >= MAX_STACK_SIZE - 1) {
        printf("\n STACK is overflow\n");
    } else {
        top++;
        stack[top] = value;
    }
}

int pop() {
    if (top <= -1) {
        printf("\n Stack is underflow\n");
        return -1;
    } else {
        return stack[top--];
    }
}

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

Output Screenshots:



```
 Enter the size of STACK[MAX=100]:10

         1.PUSH
         2.POP
         3.DISPLAY
         4.EXIT
 Enter the Choice:1
 Enter a value to be pushed:20

 Enter the Choice:1
 Enter a value to be pushed:40

 Enter the Choice:1
 Enter a value to be pushed:50

 Enter the Choice:2

         The popped elements is 50
 Enter the Choice:3

 The elements in STACK

40
20
 Press Next Choice
 Enter the Choice:4

         EXIT POINT

...Program finished with exit code 0
Press ENTER to exit console.
```

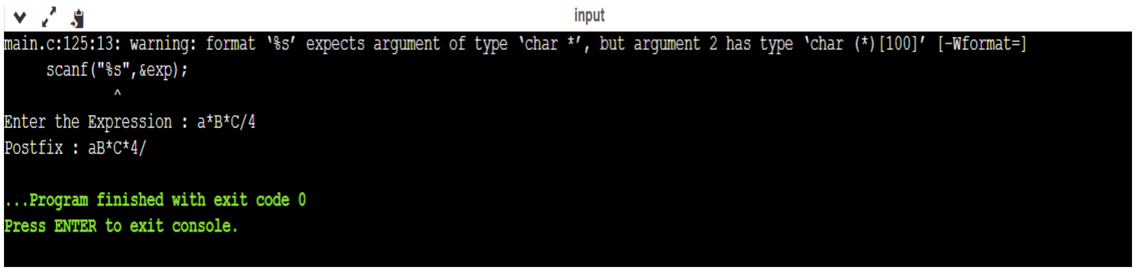
Applications of Stack:

Infix to Postfix conversion:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct stack //Declaring Stack Structure

{

    int size;

    int top;

    char *arr;

};


int stackTop(struct stack* sp){ //Function returns top of the Stack

    return sp->arr[sp->top];

}


int isEmpty(struct stack *ptr) //Function checking if stack is empty

{

    if (ptr->top == -1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}


int isFull(struct stack *ptr) // Function Checking if Stack is full

{
    if (ptr->top == ptr->size - 1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}


void push(struct stack* ptr, char val){ //Function for pushing elements to stack

    if(isFull(ptr)){

        printf("Stack Overflow! Cannot push %d to the stack\n", val);

    }

    else{

        ptr->top++;

        ptr->arr[ptr->top] = val;

    }

}


char pop(struct stack* ptr){ //Function to pop elements from the stack

    if(isEmpty(ptr)){

        printf("Stack Underflow! Cannot pop from the stack\n");

        return -1;

    }

    else{

        char val = ptr->arr[ptr->top];

        ptr->top--;

        return val;

    }

}
```

```
int precedence(char ch){ //Function for checking precedence

    if(ch == '*' || ch=='/')

        return 2;

    else if(ch == '+' || ch=='-')

        return 1;

    else

        return 0;

}


int isOperator(char ch){ //Function for checking operator

    if(ch=='+' || ch=='-' ||ch=='*' || ch=='/')

        return 1;

    else

        return 0;

}


//Infix to postfix conversion

char* infixToPostfix(char* infix){

    struct stack * sp = (struct stack *) malloc(sizeof(struct stack));

    sp->size = 10;

    sp->top = -1;

    sp->arr = (char *) malloc(sp->size * sizeof(char));

    char * postfix = (char *) malloc((strlen(infix)+1) * sizeof(char));

    int i=0; // Track infix traversal

    int j = 0; // Track postfix addition

    while (infix[i]!='\0')

    {

        if(!isOperator(infix[i])){

            postfix[j] = infix[i];

            j++;

            i++;
        }
        else{
            if(precedence(infix[i])> precedence(stackTop(sp))){
                push(sp, infix[i]);
                i++;
            }
            else{
                postfix[j] = pop(sp);
                j++;
            }
        }
    }
    while (!isEmpty(sp))
    {
        postfix[j] = pop(sp);
        j++;
    }
    postfix[j] = '\0';
    return postfix;
}
int main()
{
    char exp[100];
    char *infix;
    printf("Enter the Expression : ");
    scanf("%s",&exp);
    infix = exp;
    printf("Postfix : %s", infixToPostfix(infix));
    return 0;
}
```

OUTPUT:



```
main.c:125:13: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'char (*)[100]' [-Wformat=]
     scanf("%s",&exp);
             ^
Enter the Expression : a*B*C/4
Postfix : aB*C*4/

...Program finished with exit code 0
Press ENTER to exit console.
```

Explain the Importance of the approach followed by you

In **postfix** , as the name suggests, the operator is placed afterwards For example, if an expression is written as A+B in infix notation, the same expression can be written as AB+ in postfix notation. The order of evaluation of a postfix expression is always from left to right. Even brackets cannot alter the order of evaluation.

The expression A + B * C can be written as => AB+C* in the postfix notation

Conclusion:-

The aim of the experiment is verified.

PostLab Questions:

1) Explain how Stacks can be used in Backtracking algorithms with example.

- Consider the N-Queens problem for an example. The solution of this problem is that N queens should be positioned on a chessboard so that none of the queens can attack another queen. In the generalized N-Queens problem, N represents the number of rows and columns of the board and the number of queens which must be placed in safe positions on the board.
- The basic strategy we will use to solve this problem is to use backtracking. In this particular case, backtracking means we will perform a safe move for a queen at the time we make the move. Later, however, we find that we are stuck and there is no safe movement for the next Queen, whom we are trying to put on the blackboard, so we have to go back.
- This means we return to the queen's previous move, undo this step, and continue to search for a safe place to place this queen.

- We will use the stack to remember where we placed queens on the board, and if we need to make a backup, the queen at the top of the stack will be popped, and we will use the position of that queen to resume the search for next safe location.
- Hence, stacks can be used for the allocation of memory for most of the backtracking problems.

2) Illustrate the concept of Call stack in Recursion.

a. In recursion, the goal—the end condition—is always the **base case**.
b. It's the bottom of the barrel, the end of the recursive calls. If a recursive method calls itself, it has to stop when it hits the base case and work its way back up to the original method.
c. Each recursive function calls itself until hitting the base. Then, when the case becomes true, the result floats back to the top of the **call stack**, a list in the computer that keeps track of all the functions called in a program.
d. The fact that programming languages have call stacks is hugely important to recursive functions.

Conclusion: