**Experiment No.1**

**Title:** Implementation of Stack using C

**Problem Statement:** Write a menu driven program in C to perform following operations on the stack.

* Push
* Pop

**Algorithm:**

**Step 1** - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

**Step 2 -**Declare all the **functions** used in stack implementation.

**Step 3 -**Create a one dimensional array with fixed size (**int stack[SIZE]**)

**Step 4 -**Define a integer variable **'top'** and initialize with **'-1'**. (**int top = -1**)

**Step 5 -**In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

**push()** - Inserting value into the stack

**Step 1** -Check whether **stack** is **FULL**. (**top == SIZE-1**)

**Step 2** -If it is **FULL**, then display **"Stack is FULL!!! Stack Overflow!!!"** and terminate the function.

**Step 3** -If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

**pop()** - Delete a value from the Stack

**Step 1 -**Check whether **stack** is **EMPTY**. (**top == -1**)

**Step 2 -**If it is **EMPTY**, then display **"Stack is EMPTY!!! Stack Underflow!!!"** and terminate the function.

**Step 3 -**If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**).

**display()** - Displays the elements of a Stack

**Step 1 -**Check whether **stack** is **EMPTY**. (**top == -1**)

**Step 2 -**If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.

**Step 3 -**If it is **NOT EMPTY**, then define a variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).

**Step 4 -**Repeat above step until **i** value becomes '0'.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 3

void push();

void pop();

void display();

int stack[4];

int top=-1;

int main()

{

int ch;

do{

printf("|||Stack operations|||\n");

printf("please enter the operation\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");

scanf("%d",&ch);

switch(ch)

{

case 1:push();

break;

case 2:pop();

break;

case 3:display();

break;

case 4:exit(0);

break;

}

}while(ch!=4);

}

void push()

{

int ele;

if(top==MAX)

{

printf("\nStack is full!!!Stack Overflow!!!\n");

}

else

{

top++;

printf("Enter the element to be pushed: ");

scanf("%d",&ele);

stack[top]=ele;

printf("the element %d is pushed into the stack\n",stack[top]);

}

}

void pop()

{

if(top==-1)

{

printf("Stack is empty!!!Stack Underflow!!!\n");

}

else

{

printf("element in the stack is popped\n");

top--;

}

}

void display()

{

int i;

printf("The updated stack is:");

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

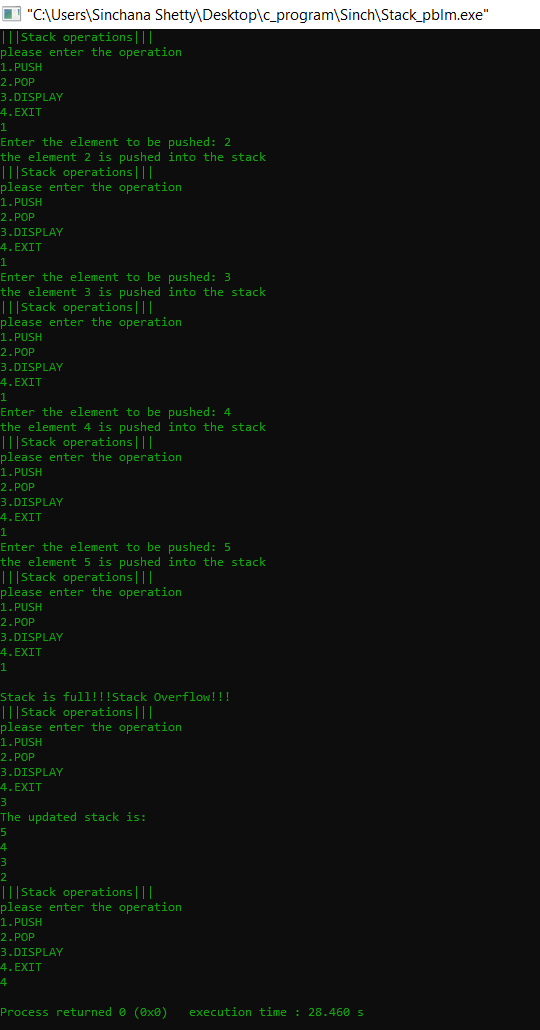
{

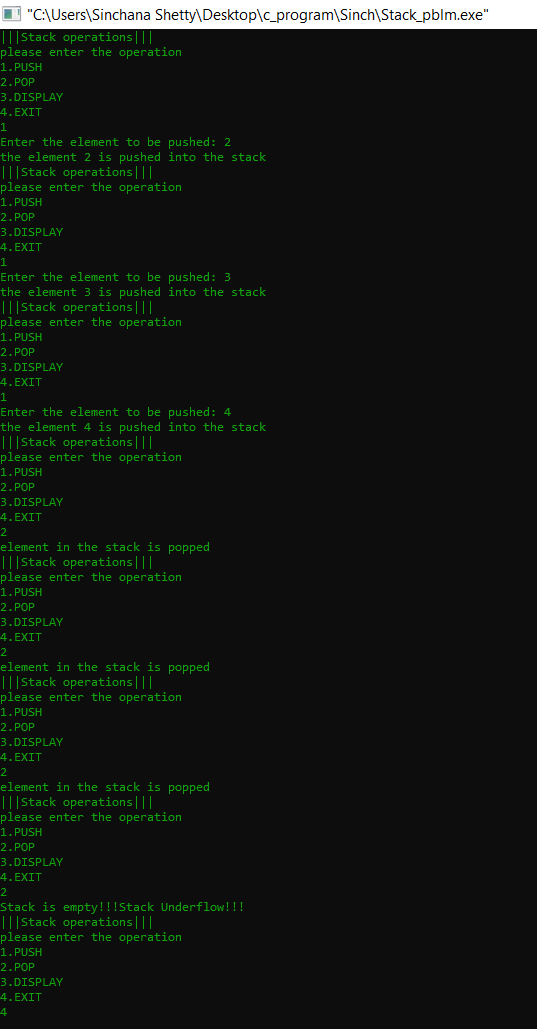
printf("\n%d",stack[i]);

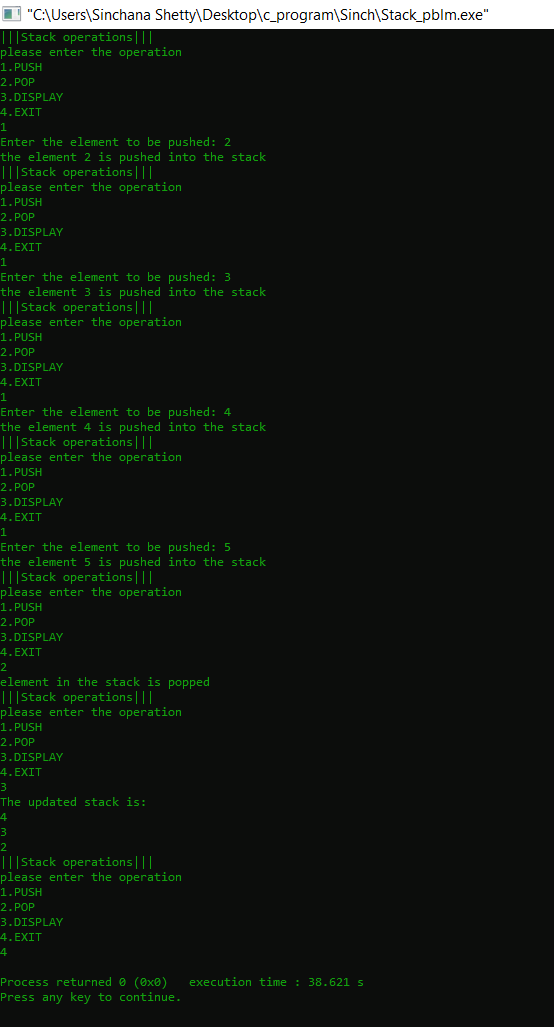
}

}

**Results:**







**Analysis(Limitations):**

Using array to implement stack is easy, it has limitation that the stack cannot go beyond a fixed size (of array). The size of Array is 100, if we just need 10 elements stack, then 90 elements memory will be wasted. In other case, if we want a stack with 500 element, we will have to change MAX value, which means program need to be compiled again.