**Experiment No. 1**

**Title :** Implementation of Stack Using C.

**Problem Statement :** Implementing linear data structure stack using array with functions

Push()

Pop()

Display()

isFull()

isEmpty()

Algorithm:

a.Algorithm for PUSH operation:

* **Step 1** − Checks if the stack is full.
* **Step 2** − If the stack is full, It produces a message called “STACK IS FULL”.
* **Step 3** − If the stack is not full, increments **top** to point next empty space.
* **Step 4** − Adds data element to the stack location, where top is pointing.
* **Step 5** − Returns success.

b. Algorithm for POP operation:

* **Step 1** − Checks if the stack is empty.
* **Step 2** − If the stack is empty, produces a message called “STACK IS EMPTY”.
* **Step 3** − If the stack is not empty, accesses the data element at which **top** is pointing.
* **Step 4** − Decreases the value of top by 1.
* **Step 5** − Returns success.

CODE:

#include<stdio.h>

# define MAX 5

int stack[5];

int top = -1;

int i,ch;

int isFull()

{

if(top == MAX-1)

{

printf("Stack is FULL\n");

return(1);

}

else

{

printf("Stack is NOT FULL\n");

return(0);

}

}

int isEmpty()

{

if(top == -1)

{

printf("Stack is EMPTY\n");

return(1);

}

else

{

printf("Stack is NOT EMPTY\n");

return(0);

}

}

void push()

{

int var;

if(isFull()==0)

{

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

scanf("%d",&var);

top = top + 1;

stack[top] = var;

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

}

}

void pop()

{

if(isEmpty()==0)

{

printf("%d is popped\n",stack[top]);

top = top - 1;

}

}

void display()

{

for(i=top;i>-1;--i)

{

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

}

}

int main()

{

start :

printf("Enter your choice :\n1.PUSH\t2.POP\t3.ISEMPTY\t4.ISFULL\t5.DISPLAY\nchoice : ");

scanf("%d",&ch);

switch(ch)

{

case 1 :

push();

break;

case 2 :

pop();

break;

case 3 :

isempty();

break;

case 4 :

isfull();

break;

case 5 :

display();

break;

default :

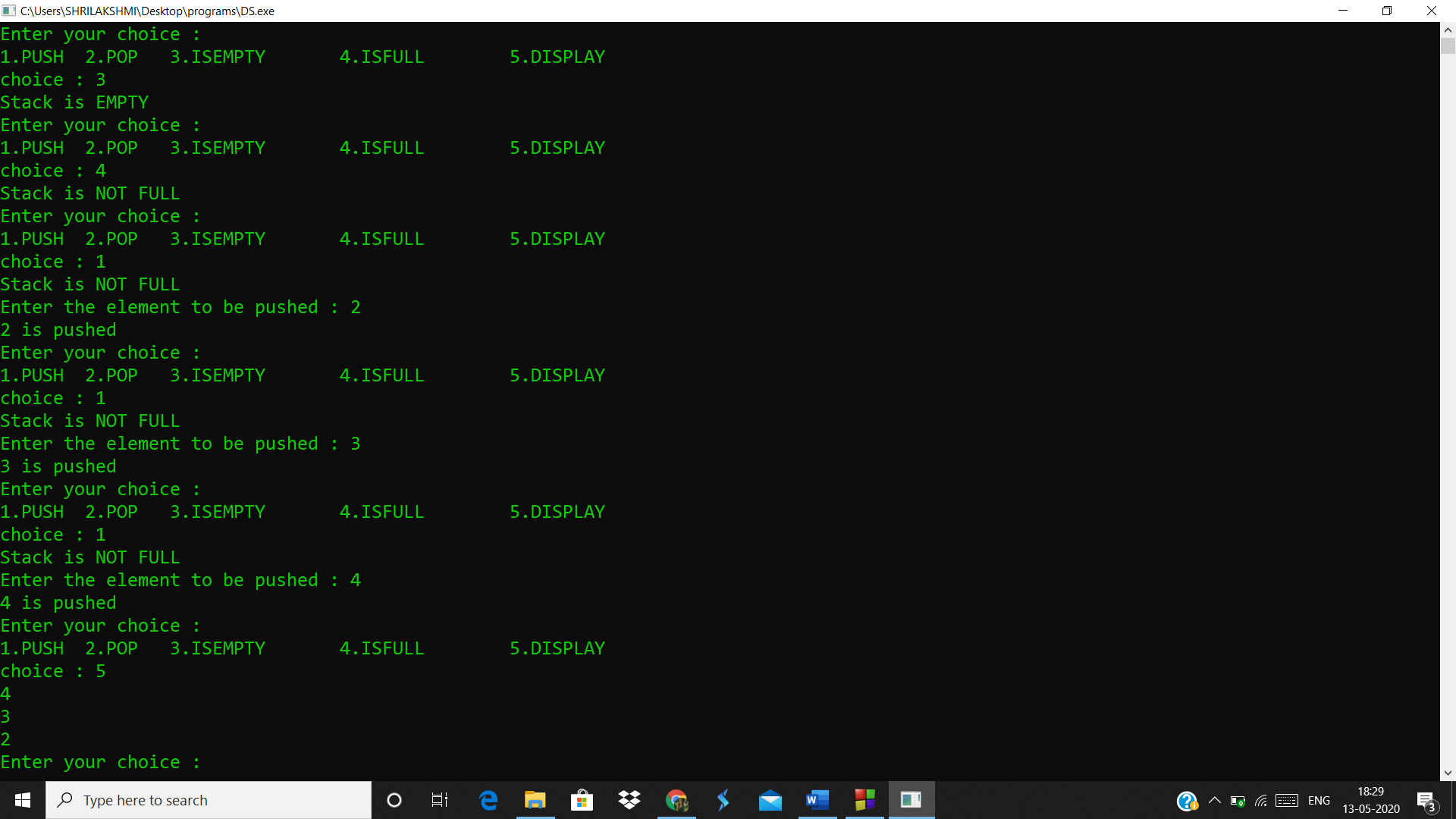
printf("Enter the correct choice\n");

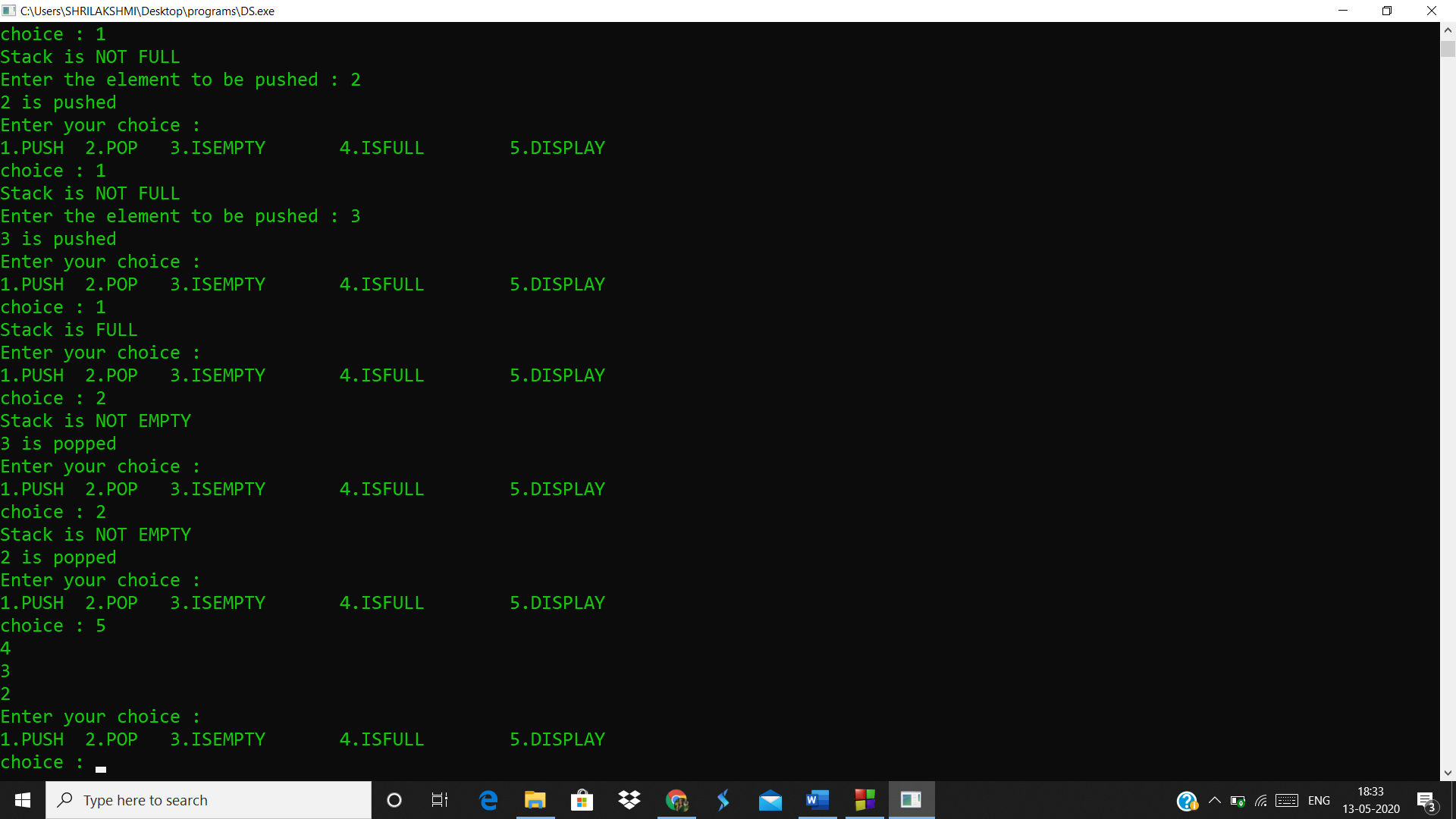
break;

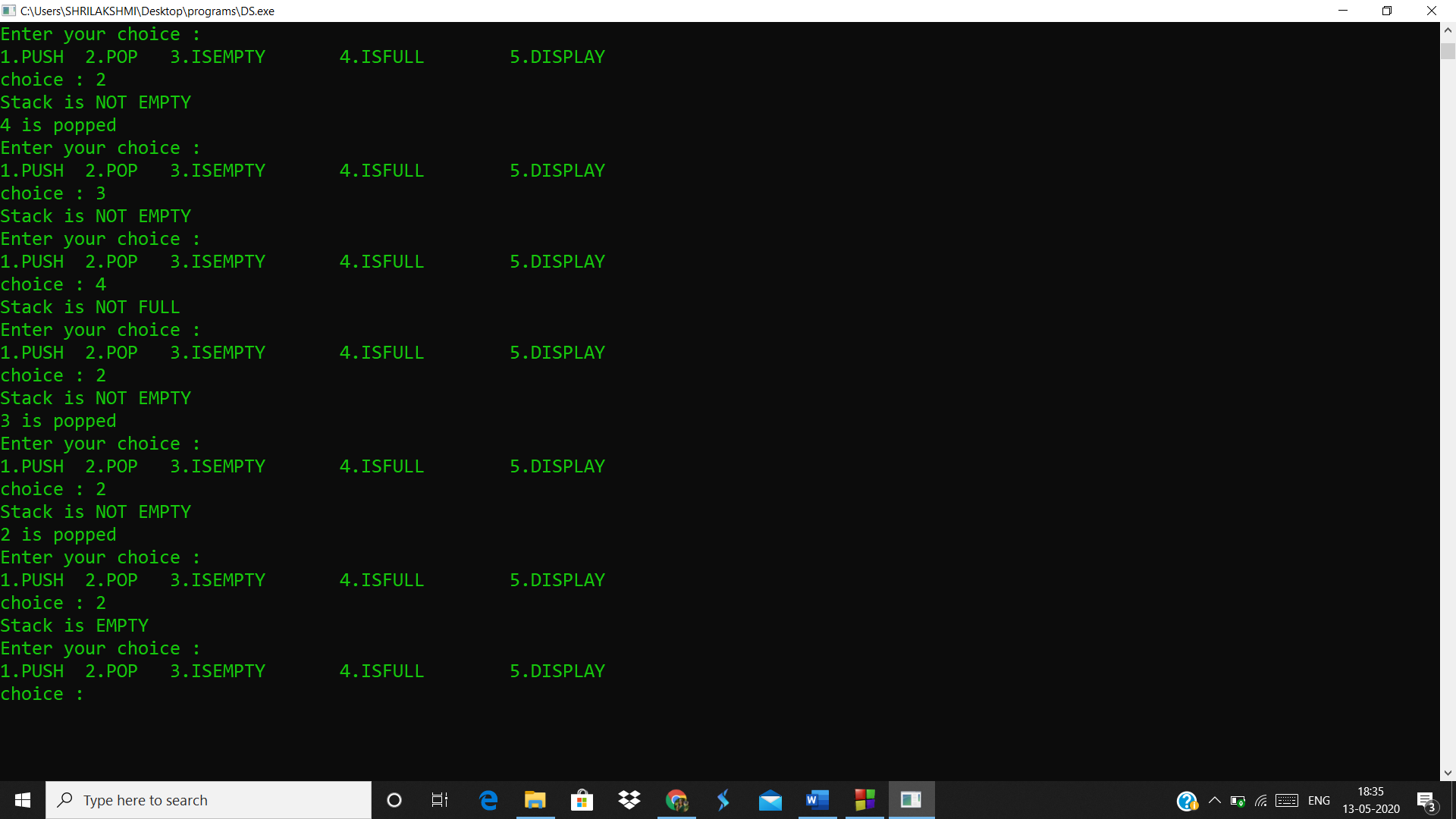
}

goto start;

}





j

Analysis(Limitations):

* Inbetween elements cannot be accessed i-e only top element can be accessed.
* Stack memory is very limited.
* Creating too many objects on the stack can increase the risk of stack overflow.