STACK OPERATIONS

Aim: To implement operations of stack using array.

Theory: Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named **top**. The **top** pointer provides top value of the stack without actually removing it.

Program:

```
#include <stdio.h>                    //preprocessor directives
```

```
#include <stdlib.h>
#include <string.h>
#define MAX 100                                   //defining array size
int stack[100],top=-1;                            //initializing stack and top=-1
   void push(int ele)                             //called function for push
   {
      if(top==MAX-1)                              //conditional stmt to check stack
      {
         printf("stack is full");
         return ;
      }
      top++;                                      //incrementing top by 1
      stack[top]=ele;                             //inserting element at top
      printf("element pushed is%d",stack[top]);  //prints output
   }
int pop()                                         //called function for pop
{
   if(top==-1)                                    //conditional statement to check stack
   {
      printf("stack is empty");
      return -1;                                  //returns -1
   }
   int ele=stack[top];                            //element to be popped
   top--;                                         //decrementing top by 1
```

```
   printf("element popped is:%d",ele);    //prints output
}
void size()                          //called function for size
{
   printf("size of stack is %d",top+1);      //returns stack size
}
void display()                                //called function for display
{
   int i;                                  //initializing variable i
   printf("elements are:");
   for(i=0;i<=top;i++)                  //looping condition
   printf("%d",stack[i]);                     //prints elements of stack
}
void main()                                   //main function
{
   int choice,ele;
   printf("1.push\n2.pop\n3.size\n4.display\n5.exit");
   while(1)                          //looping statement if true
   {
      printf("enter your choice");
      scanf("%d",&choice);                      //reads choice
      switch(choice)                //selection statement based on choice
      {
         case 1:printf("enter the element");         //enters value
```

```
        scanf("%d",&ele);
        push(ele);                    //function call for push
        break;                        //comes  out of switch condition
        case 2:pop();                 //function call for pop
        break;                        //comes out of switch condition
        case 3:size();                //function call for size
        break;                        //comes out of switch condition
        case 4:display();             //function call for display
        break;                        //comes out of switch condition
        case 5:
        exit (0);
      }
   }
}
```

Algorithm:

Push operation:

Step 1: Initialize variable x

Step 2: Let top=-1, 'N' be size of array

Step 3: if top=N-1

then stack overflow

else

top=top+1

stack[top]=x

end if

Step 4: Exit

Pop operation:

Step 1: Initialize variable x

Step 2: Let top=-1, ‘N’ be size of array

Step 3: if top=-1

then stack underflow

else

x=stack[top]

top=top-1

end if

Step 4: Exit

Display operation:

Step 1: Initialize variable x

Step 2: Let top=-1, ‘N’ be size of array

Step 3: if top=-1

then stack underflow

else

for i=0 to N-1

print arr[i]

end for

end if

Step 4: Exit

Size operation:

Step 1: Initialize variable x

Step 2: Let top=-1, ‘N’ be size of array

Step 3: if top=-1

then stack underflow

else

print top+1

end if

Step 4: Exit

Peek operation:

Step 1: Initialize variable x

Step 2: Let top=-1, ‘N’ be size of array

Step 3: if top=-1

then stack underflow

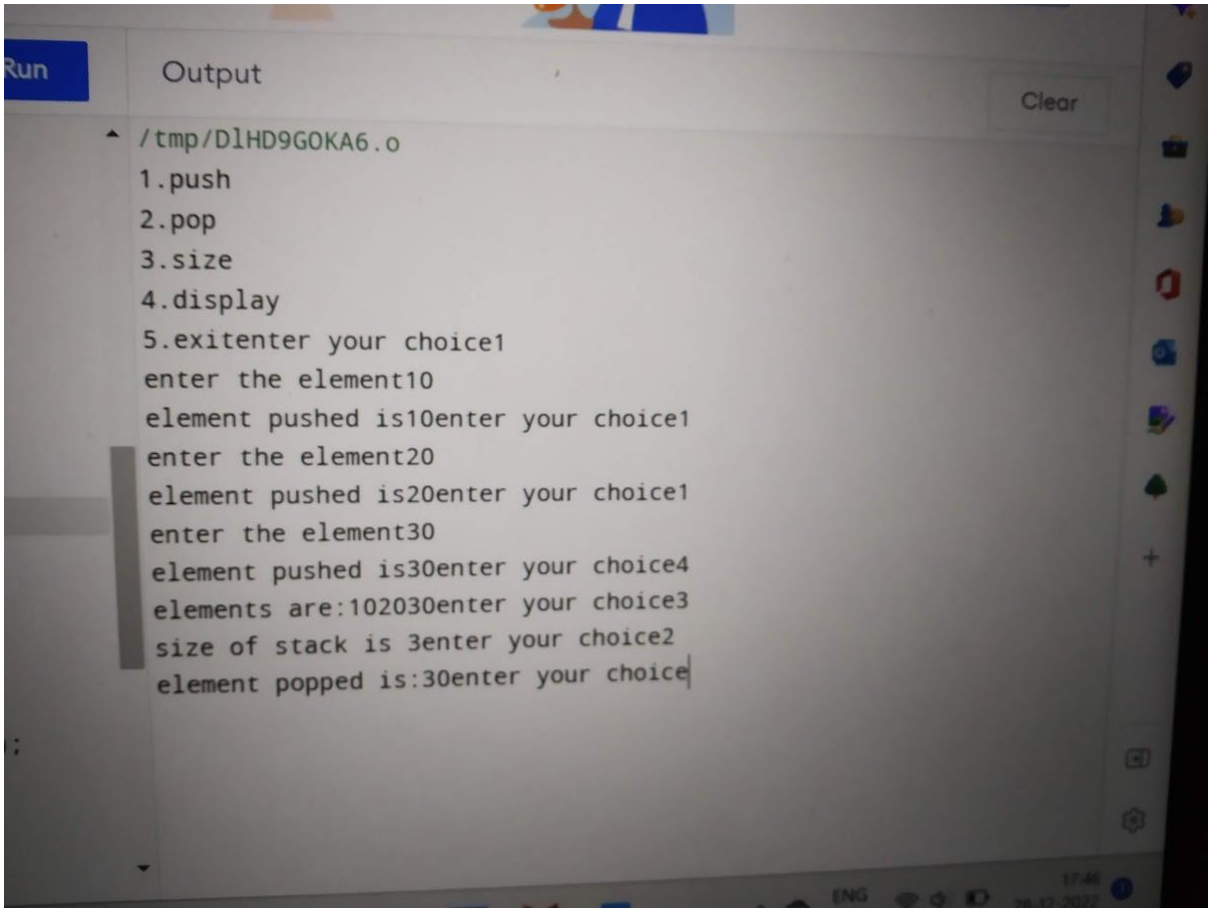
else

print stack[top]

end if

Step 4: Exit

Output:


Run
Output
Clear
/tmp/DlHD9GOKA6.o
1.push
2.pop
3.size
4.display
5.exitenter your choice1
enter the element10
element pushed is10enter your choice1
enter the element20
element pushed is20enter your choice1
enter the element30
element pushed is30enter your choice4
elements are:102030enter your choice3
size of stack is 3enter your choice2
element popped is:30enter your choice