Course Name:	Programming in C	Semester:	II
Date of Performance:	09/ 04/ 2025	DIV/ Batch No:	C22
Student Name:	Ashwera Hasan	Roll No:	16010124107

Experiment No: 9

Title: Implementation of Stack Data Structure Using Arrays

Aim and Objective of the Experiment:

Write a program in C to implement the stack data structure using arrays and perform basic stack operations such as push, pop, peek, and display.

COs to be achieved:

CO: Apply basic concepts of C programming for problem-solving.(CO1 and CO2), file handling (CO5)

Theory:

A stack is a linear data structure that follows the **Last In, First Out (LIFO)** principle. It allows insertion (push) and deletion (pop) operations at one end, called the **top** of the stack.

1. **Basic Stack Operations:**
   - **Push(x):** Adds an element `x` to the top of the stack.
   - **Pop():** Removes and returns the top element from the stack.
   - **Peek():** Returns the top element without removing it.
   - **isEmpty():** Checks if the stack is empty.
   - **isFull():** Checks if the stack is full (in case of an array implementation).
   - **Display():** Shows all elements in the stack.
2. **Applications of Stack:**
   - Expression evaluation (infix to postfix conversion, postfix evaluation)
   - Function call management (recursion)
   - Undo/Redo operations in applications
   - Backtracking algorithms (Maze solving, Depth First Search)

Procedure:

1. **Initialize the Stack:**
   - Define an array of fixed size.
   - Initialize `top = -1` (indicating an empty stack).

2. **Implement Push Operation:**
   - Check if the stack is full (`top == size-1`).
   - If not, increment `top` and insert the element.
3. **Implement Pop Operation:**
   - Check if the stack is empty (`top == -1`).
   - If not, remove and return the top element, then decrement `top`.
4. **Implement Peek Operation:**
   - Return the element at the `top` without removing it.
5. **Implement Display Operation:**
   - Print all elements from `top` to `0`.
6. **Test the Stack Implementation:**
   - Perform multiple push and pop operations.
   - Validate the expected outputs.

Problem Statements:

Develop a program to implement a stack data structure using arrays. The program should support the following operations:

1. **Push(x):** Insert an element `x` into the stack.
2. **Pop():** Remove and return the top element from the stack.
3. **Peek():** Display the top element without removing it.
4. **isEmpty():** Check whether the stack is empty.
5. **isFull():** Check whether the stack is full.
6. **Display():** Print all elements present in the stack.

Constraints:

- The stack should be implemented using a fixed-size array.
- The stack follows the **Last In, First Out (LIFO)** principle.
- The program should handle cases of **stack overflow** (pushing into a full stack) and **stack underflow** (popping from an empty stack).

Input/Output Format:

- **Input:** The user should be able to select operations and enter values accordingly.
- **Output:** Display the stack status after each operation, including error messages if applicable.

Enter stack size: 5
Choose operation: 1-Push, 2-Pop, 3-Peek, 4-Display, 5-Exit
1
Enter value to push: 10
1
Enter value to push: 20
4

```
Stack elements: [10, 20]
3
Top element: 20
2
Popped element: 20
4
Stack elements: [10]
5
Exiting program...
```

Code :

```c
#include <stdio.h>
int main()
{

   int size;
   printf("define size of array\n");
   scanf("%d",&size);
   int arr[size];
   int top = -1;
   printf("Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.\nEnter -1 to exit. \n");
   int choice;
   scanf("%d",&choice);
   while(choice!=-1)
   {
      if(choice>4||choice<-1){
         printf("Invalid choice.\n");
      }
      if(choice==1)
   {if(top == size-1)//push
   {
      printf("Stack full\n");
   }
   else{
         printf("Enter element: \n");
      top++;
      scanf("%d",&arr[top]);
   }}
   else if (choice==2)
   {if(top==-1)//pop
   {
      printf("Stack empty\n");
   }
   else{
```

```
    {
        printf("Stack empty\n");
    }
    else{
        printf("Popped out %d\n",arr[top]);
        top--;
    }}
    else if(choice==3){if(top==-1)//peek
    {
        printf("Stack empty\n");
    }
    else{
        printf("Peeked at %d\n",arr[top]);
    }}
    else{
    int i;
    for(i=top;i>=0;i--)
    {
        printf("[%d],",arr[i]);
    }
    printf("\n");
    }
    printf("Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.\nEnter -1 to exit. \n");

    scanf("%d",&choice);
    }
}
```

Output:

```
C:\Users\Student\Desktop\sta
define size of array
3
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
1
Enter element:
1
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
1
Enter element:
2
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
1
Enter element:
3
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
3
Peeked at 3
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
4
[3],[2],[1],
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
2
Popped out 3
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
4
[2],[1],
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
8
Invalid choice.
[2],[1],
Enter 1 to push, 2 to pop, 3 to peek, and 4 to print all numbers in stack.
Enter -1 to exit.
-1

Process returned -1 (0xFFFFFFFF)   execution time : 24.088 s
Press any key to continue.
```

Conclusion:

This experiment implements a basic stack using an array, allowing users to push, pop, peek, and print elements. It handles stack overflow/underflow and invalid inputs. The loop runs until the user exits by entering -1. The code demonstrates fundamental stack operations with clear prompts and error checks, making it functional and user-friendly.

Signature of faculty in-charge with Date: