**Question**

**Write a C program to implement a stack using array, implementation including the function to check whether the stack is empty. Push an element into the stack; pop an element from a stack. Return the top element from the stack, display the stack elements.**

**Code**

*/\*\**

*\* Write a C program to implement a stack using array, implementation including the function to check whether the stack is empty.*

*\* Push an element into the stack; pop an element from a stack. Return the top element from the stack, display the stack elements.*

*\*/*

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>     *// For INT\_MIN*

#define SIZE 100

*// Create a stack with capacity of 100 elements*

int stack[SIZE];

*// Initially stack is empty*

int top = -1;

*/\* Function declaration to perform push and pop on stack \*/*

void push(int element);

int  pop();

void isEmpty();

int peek();

void display();

int main()

{

    int choice, data;

    while(1)

    {

*/\* Menu \*/*

        printf("------------------------------------\n");

        printf("    STACK IMPLEMENTATION PROGRAM    \n");

        printf("------------------------------------\n");

        printf("1. Push\n");

        printf("2. Pop\n");

        printf("3. Peek\n");

        printf("4. Size\n");

        printf("5. Check whether empty\n");

        printf("6. Display\n");

        printf("7. Exit\n");

        printf("------------------------------------\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

                printf("Enter data to push into stack: ");

                scanf("%d", &data);

*// Push element to stack*

                push(data);

                break;

            case 2:

                data = pop();

*// If stack is not empty*

                if (data != INT\_MIN)

                    printf("Data => %d\n", data);

                break;

            case 3:

                data = peek();

*// If stack is not empty*

                if (data != INT\_MIN)

                    printf("Data on top of stack => %d\n", data);

                break;

            case 4:

                printf("Stack size: %d\n", top + 1);

                break;

            case 5:

                isEmpty();

                break;

            case 6:

                display();

                break;

            case 7:

                printf("Exiting from app.\n");

                exit(0);

                break;

            default:

                printf("Invalid choice, please try again.\n");

        }

        printf("\n\n");

    }

    return 0;

}

*/\*\**

*\* Functiont to push a new element in stack.*

*\*/*

void push(int element)

{

*// Check stack overflow*

    if (top >= SIZE)

    {

        printf("Stack Overflow, can't add more element element to stack.\n");

        return;

    }

*// Increase element count in stack*

    top++;

*// Push element in stack*

    stack[top] = element;

    printf("Data pushed to stack.\n");

}

*/\*\**

*\* Function to pop element from top of stack.*

*\*/*

int pop()

{

*// Check stack underflow*

    if (top < 0)

    {

        printf("Stack is empty.\n");

*// Throw empty stack error/exception*

*// Since C does not have concept of exception*

*// Hence return minimum integer value as error value*

*// Later in code check if return value is INT\_MIN, then*

*// stack is empty*

        return INT\_MIN;

    }

*// Return stack top and decrease element count in stack*

    return stack[top--];

}

void isEmpty(){

    if (top < 0)

    {

        printf("Stack is empty.\n");

    }

    else{

        printf("Stack is not empty.\n");

    }

}

int peek(){

*// Check stack underflow*

    if (top < 0)

    {

        printf("Stack is empty.\n");

*// Throw empty stack error/exception*

*// Since C does not have concept of exception*

*// Hence return minimum integer value as error value*

*// Later in code check if return value is INT\_MIN, then*

*// stack is empty*

        return INT\_MIN;

    }

*// Return stack top*

    return stack[top];

}

void display(){

    int l = top + 1;

    printf("Elements on stack:\n");

    for(int i = 0; i < l; i++){

        printf("%d\n", stack[i]);

    }

}

**Output**



**Question**

**Write a C program to implement a stack using linkedlist, implementation including the function to check whether the stack is empty. \* Push an element into the stack; pop an element from a stack. Return the top element from the stack, display the stack elements.**

**Code**

*/\*\**

*\* Write a C program to implement a stack using linkedlist, implementation including the function to check whether the stack is empty.*

*\* Push an element into the stack; pop an element from a stack. Return the top element from the stack, display the stack elements.*

*\*/*

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>     *// For INT\_MIN*

#define CAPACITY 10000  *// Stack maximum capacity*

*// Define stack node structure*

struct stack

{

    int data;

    struct stack \*next;

} \*top;

*// Stack size*

int size = 0;

*/\* Function declaration to perform push and pop on stack \*/*

void push(int element);

int  pop();

int main()

{

    int choice, data;

    while(1)

    {

*/\* Menu \*/*

        printf("------------------------------------\n");

        printf("    STACK IMPLEMENTATION PROGRAM    \n");

        printf("------------------------------------\n");

        printf("1. Push\n");

        printf("2. Pop\n");

        printf("3. Peek\n");

        printf("4. Size\n");

        printf("5. Check whether empty\n");

        printf("6. Display\n");

        printf("7. Exit\n");

        printf("------------------------------------\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

                printf("Enter data to push into stack: ");

                scanf("%d", &data);

*// Push element to stack*

                push(data);

                break;

            case 2:

                data = pop();

*// If stack is not empty*

                if (data != INT\_MIN)

                    printf("Data => %d\n", data);

                break;

            case 3:

                data = peek();

*// If stack is not empty*

                if (data != INT\_MIN)

                    printf("Data on top of stack => %d\n", data);

                break;

            case 4:

                printf("Stack size: %d\n", size);

                break;

            case 5:

                isEmpty();

                break;

            case 6:

                display();

                break;

            case 7:

                printf("Exiting from app.\n");

                exit(0);

                break;

            default:

                printf("Invalid choice, please try again.\n");

        }

        printf("\n\n");

    }

    return 0;

}

*/\*\**

*\* Functiont to push a new element in stack.*

*\*/*

void push(int element)

{

*// Check stack overflow*

    if (size >= CAPACITY)

    {

        printf("Stack Overflow, can't add more element to stack.\n");

        return;

    }

*// Create a new node and push to stack*

    struct stack \* newNode = (struct stack \*) malloc(sizeof(struct stack));

*// Assign data to new node in stack*

    newNode->data = element;

*// Next element after new node should be current top element*

    newNode->next = top;

*// Make sure new node is always at top*

    top = newNode;

*// Increase element count in stack*

    size++;

    printf("Data pushed to stack.\n");

}

*/\*\**

*\* Function to pop element from top of stack.*

*\*/*

int pop()

{

    int data = 0;

    struct stack \* topNode;

*// Check stack underflow*

    if (size <= 0 || !top)

    {

        printf("Stack is empty.\n");

*// Throw empty stack error/exception*

*// Since C does not have concept of exception*

*// Hence return minimum integer value as error value*

*// Later in code check if return value is INT\_MIN, then*

*// stack is empty*

        return INT\_MIN;

    }

*// Copy reference of stack top to some temp variable*

*// Since we need to delete current stack top and make*

*// Stack top its next element*

    topNode = top;

*// Copy data from stack's top element*

    data = top->data;

*// Move top to its next element*

    top = top->next;

*// Delete the previous top most stack element from memory*

    free(topNode);

*// Decrement stack size*

    size--;

    return data;

}

*/\*\**

*\* Function to peek element from top of stack.*

*\*/*

int peek()

{

    int data = 0;

*// Check stack underflow*

    if (size <= 0 || !top)

    {

        printf("Stack is empty.\n");

*// Throw empty stack error/exception*

*// Since C does not have concept of exception*

*// Hence return minimum integer value as error value*

*// Later in code check if return value is INT\_MIN, then*

*// stack is empty*

        return INT\_MIN;

    }

*// Copy data from stack's top element*

    data = top->data;

    return data;

}

*/\*\**

*\* Function to check whether stack is empty or not*

*\*/*

void isEmpty(){

*// Check stack underflow*

    if (size <= 0 || !top)

    {

        printf("Stack is empty.\n");

    }

    else{

        printf("Stack is not empty.\n");

    }

}

void display(){

    if (size <= 0 || !top)

    {

        printf("Stack is empty.\n");

    }

    else{

        struct stack \* temp;

        temp = top;

        while(temp != NULL){

            printf("%d\n", temp->data);

            temp = temp->next;

        }

    }

}

**Output**

