Ex : 2

Title: Design, develop, and implement a data structure of integers that follows the LIFO principle and demonstrates its operations.

Problem Description: The program creates the data structure and performs operations like inserting an element, deleting an element, and displaying the status of the data structure.

Method: In this program, create the data structure and support the program with appropriate functions for each of the above operations. Demonstrate Overflow and Underflow situations and support the program with appropriate functions for each of the above operations. When overflow occurs increase the size of the data structure by array doubling technique.

Theory Reference: Module 1

Explanation

1. **Dynamic Memory Allocation**:
    - The array **a** is dynamically allocated using malloc(). The size of the array is determined at runtime based on user input (max).
    - The array **a** is dynamically is also reallocated using realloc() with size 2*max.
2. **Basic stack operations**

    A stack is a linear data structure that follows the Last In, First Out (LIFO) principle, meaning the last element added to the stack is the first one to be removed.

    Basic stack operations:

    Push Operation - Adds an element to the top of the stack.

    Pop Operation - Removes and returns the top element from the stack.

3. **Handling of stack overflow and underflow conditions** - Stack is also resized in case of overflow using array doubling technique.

4. **Freeing Memory**: The dynamically allocated memory is freed using free() to avoid memory leaks.

Algorithm

Step 1: Initialize

1. Set top = -1 to indicate the stack is empty.
2. Allocate memory for the stack a with size max using malloc.

Step 2: Push Operation

1. **Check if the stack is full**:
   - If top == max - 1, it means the stack is full.
   - Double the size of the stack (max = 2 * max) and resize the stack using realloc.
   - Print a message indicating the stack has been resized.
2. **Push the element**:
   - Increment top by 1.
   - Insert the element ele at position a[top].

Step 3: Pop Operation

1. **Check if the stack is empty**:
   - If top == -1, print a message indicating "stack underflow" and return -999.
2. **Pop the element**:
   - Return the element at a[top].
   - Decrement top by 1.

Step 4: Display Operation

1. **Check if the stack is empty**:
   - If top == -1, print "stack is empty."
2. **Display the stack elements**:
   - Print all elements from a[top] to a[0].

Step 5: Main Loop

1. **Initialize stack size**:
    - Ask the user to input the initial value of max and allocate memory for the stack a using malloc.
2. **Loop through the menu**:
    - Print the menu and ask the user to enter a choice.
    - Based on the user's choice, call the corresponding function:
        - **Choice 1**: Call push() to push an element onto the stack.
        - **Choice 2**: Call pop() to remove and display the top element.
        - **Choice 3**: Call display() to print all elements in the stack.
        - **Choice 4**: Exit the program and free the allocated memory.
    - If the user enters an invalid choice, print "invalid choice."

Step 6: Terminate

- When the user chooses to exit, free the allocated memory for the stack and terminate the program.