CSE108 – Computer Programming Lab

Lab 13

Stacks

30/05/2025

Imagine you are developing a simple text editor. The editor supports an *undo* feature that stores past actions (like inserting or deleting text). These actions are stored using a stack so that the most recent action can be undone first.

In a basic version of the text editor, every time the user performs an action (e.g., inserts or deletes a character), you push this action into the stack. Memory is reallocated each time an action is added.

Part 1. (5 pts) This function creates an empty stack with an initial capacity.

Part 2. (20 pts) Implement a function that pushes a new action into the stack. Each time an action is added, reallocate the memory to fit **exactly one more** element. This function takes the stack, the current action, and the size.

Part 3. (20 pts) Implement an undo() function that removes the most recent action from the stack and returns it. If the stack is empty, display a warning message.

Part 4. (10 pts) Display the current action history from **top to bottom**. Actions that have been undone should **not** be displayed.

Part 5. (5 pts) Free all dynamically allocated memory at the end of the program to prevent memory leaks.

Part 6. (30 pts) Improve the stack with better performance: start with a given capacity (e.g., 2). When the number of actions reaches capacity, **double** the size of the stack.

Part 7. (10 pts) When using the doubling capacity stack, if the number of actions falls below half of the current capacity, shrink the capacity of the stack by half to optimize memory usage. Print a message on terminal like: Stack shrunk to capacity: 2

Rules:

- Use dynamic memory only (malloc, calloc, realloc, free).
- Do not use fixed arrays or global variables.
- Store each action as a char* string.
- Maintain clean output formatting and memory management.

```
Example Output:

Which stack will you use?

1.One-by-one growth stack
2.Doubling capacity stack

Choice: 1 //Go to main menu directly

Choice: 2 //Ask the initial capacity
Enter initial capacity stack: 2

1. Perform Action
2. Undo Last Action
3. Print Action History
4. Exit

Choice: 1
Enter action: Insert A
Action recorded.

Choice: 1
Enter action: Insert B
Action recorded.

Choice: 1
Enter action: Delete C
Stack resized to capacity: 4 //If the stack choice is 2
Action recorded.

Choice: 3
Action History (Top to Bottom):
Delete C
Insert B
Insert A

Choice: 2
Undoing action: Delete C
Stack shrunk to capacity: 2  //If the stack choice is 2

Choice: 3
Action History (Top to Bottom):
Insert B
Insert A

Choice: 4
Cleaning up memory... Goodbye!
```