Institute for Advancing Intelligence, TCG CREST
(TCG Centres for Research and Education in Science and Technology)

Introduction to Programming and Data Structures
Ph.D. Coursework: First year, First Semester (Session: 2024-25)
Assignment #04

Full Marks: 300
Clarification Deadline: **2024-Sep-26**

Instructor: Dr. Laltu Sardar
Submission Deadline: **2024-Sep-29**

This is not the final problem-set. Only problem AP0401 is the final one.

Instructions

1. Errors must be handled in all possible functions used, whether from libraries or written by yourself
2. Function names and variable names should clearly describe their purpose.
3. Write the program in such a way, that program does not fails.
4. Magic numbers (like 100 in `array[100]`) should not be hard-coded across the programs. Instead define them as macros (E.g. `#define ARRAY_SIZE 100` and later `array[ARRAY_SIZE]`).

1 Problem #AP0401: Indentation

Indentation in programming is the use of spaces or tabs at the beginning of lines to structure code for readability and, in some languages like Python, it is a required syntax to define code blocks.

Indentation Rules in C

1. Indent code inside control structures like `if`, `else`, `for`, `while`, and `switch` by a fixed number of spaces or a tab.
2. Always indent code within functions and blocks of code enclosed by curly braces {}.
3. Maintain consistent indentation throughout the codebase, typically using 2, 4, or 8 spaces or a single tab.
4. Avoid mixing tabs and spaces for indentation to ensure uniformity across editors.
5. Code inside nested blocks should be indented further than outer blocks to reflect hierarchy.
6. Align the closing brace } of a block with the statement that opened the block.
7. Use indentation to visually separate declarations and statements within functions.

Problem statement: Given a list of files on the **command line**, convert them with proper indentation.

Input: Give user option to use 2,4,8 spaces or tabs. and a set of filenames from command line.

Output: The files with same name and with indentation.

[100]

2 Problem #AP0402: Stack Implementation

A stack is a linear data structure that follows the Last In First Out (LIFO) principle. The following operations are commonly performed on a stack:

- `push(int data)` The `push` operation adds an element to the top of the stack.
  - Example: If the stack is initially {3, 7, 2} and we push the value 5, the stack becomes {3, 7, 2, 5}.
- `pop()` The `pop` operation removes and returns the top element of the stack. If the stack is empty, an error is usually thrown.
  - Example: If the stack is {3, 7, 2, 5} and we pop, the stack becomes {3, 7, 2} and the returned value is 5.
- `peek()` The `peek` operation returns the top element of the stack without removing it.
  - Example: If the stack is {3, 7, 2, 5} and we peek, the returned value is 5, but the stack remains {3, 7, 2, 5}.
- `isEmpty()` The `isEmpty` operation checks if the stack contains any elements. It returns `true` if the stack is empty, and `false` otherwise.
  - Example: If the stack is {3, 7, 2}, `isEmpty()` returns `false`. If the stack is {}, `isEmpty()` returns `true`.
- `isFull()` The `isFull` operation checks if the stack is full (applicable for stacks implemented using a fixed-size array). It returns `true` if the stack cannot hold any more elements.
  - Example: If the stack size is fixed at 5 and the stack is {3, 7, 2, 5, 8}, `isFull()` returns `true`.

Problem Statement

Implement stack using linked list and array.

Input

Take only from the terminal. Provide the user options to choose operations from the list. Then, take the required input from the user accordingly.

Output

Display the stack after each operation. [100]

3 Problem #AP0403: Queue Implementation

A queue is a linear data structure that follows the First In First Out (FIFO) principle. The following operations are commonly performed on a queue:

- `enqueue(int data)` The `enqueue` operation adds an element to the rear (end) of the queue.
  - Example: If the queue is initially {10, 20, 30} and we enqueue 40, the queue becomes {10, 20, 30, 40}.
- `dequeue()` The `dequeue` operation removes and returns the front element from the queue. If the queue is empty, an error is usually thrown.
  - Example: If the queue is {10, 20, 30, 40} and we dequeue, the queue becomes {20, 30, 40} and the returned value is 10.
- `front()` The `front` operation returns the front element of the queue without removing it.
  - Example: If the queue is {10, 20, 30, 40} and we call `front()`, the returned value is 10, and the queue remains {10, 20, 30, 40}.
- `isEmpty()` The `isEmpty` operation checks if the queue contains any elements. It returns `true` if the queue is empty, and `false` otherwise.
  - Example: If the queue is {10, 20}, `isEmpty()` returns `false`. If the queue is {}, `isEmpty()` returns `true`.
- `isFull()` The `isFull` operation checks if the queue is full (applicable for queues implemented using a fixed-size array). It returns `true` if the queue cannot hold any more elements.
  - Example: If the queue size is fixed at 4 and the queue is {10, 20, 30, 40}, `isFull()` returns `true`.

Problem Statement

Implement queue using linked list and array.

Input

Take only from the terminal. Provide the user options to choose operations from the list. Then, take the required input from the user accordingly.

Output

Display the queue after each operation.

[100]