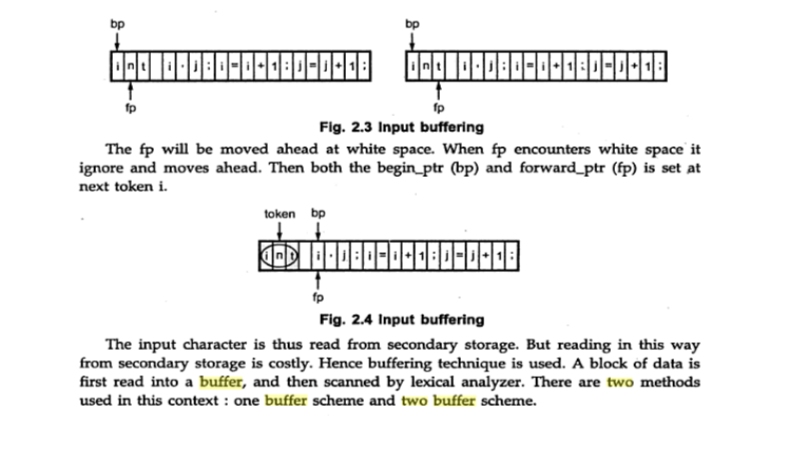
**Compiler Construction Lab CS - 471L**

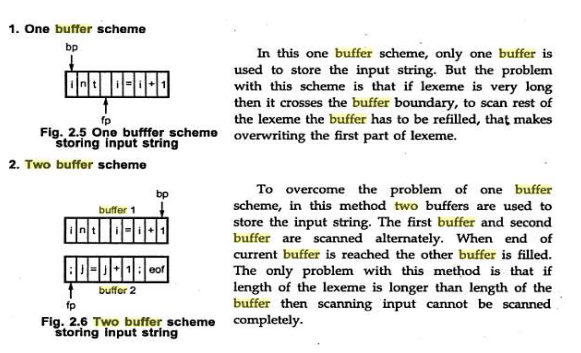
**Lab Manual – Week 2**

**Lab Assignment: Implementing User Input Source Code Buffering with Two Buffers in Python with multithreading. (Double Buffering)**

**Introduction**: In this lab assignment, you will enhance the user input source code buffering system by introducing multithreading. Multithreading allows you to achieve concurrent execution, making your program more responsive and efficient when handling user input.

An input buffering scheme using two buffers, often referred to as a "double-buffering" or "ping-pong" buffering scheme, works by maintaining two separate buffers to efficiently read and process incoming data. This scheme is commonly used in scenarios where data is continuously arriving, and you want to ensure smooth and uninterrupted processing without waiting for new data to arrive. Here's how it typically works:





**Initialization:**

Two buffers, usually of equal size, are created. Let's call them Buffer A and Buffer B.

A pointer or index is maintained for each buffer to keep track of the current position in that buffer.

**Filling the Active Buffer (Ping):**

Initially, one of the buffers (e.g., Buffer A) is designated as the "active" buffer. Data is read from the source (e.g., a file or user input) and stored in the active buffer. The data is read into the buffer until it's full, or a certain amount of data has been received.

**Processing Data from the Active Buffer:**

While one buffer is actively receiving data, the other buffer (e.g., Buffer B) is available for processing. The program can read and process data from the available buffer (Buffer B) without interruption.

**Switching Buffers (Pong):**

When the active buffer is full or a predefined condition is met, the roles of the buffers are switched. The active buffer becomes the available buffer, and vice versa. The data in the newly active buffer is read from the source while the program continues to process data from the previously active buffer.

**Continuous Operation:**

This process of "ping-pong" buffering continues in a continuous loop. As one buffer is filled, the program processes data from the other buffer. Switching between buffers occurs seamlessly, ensuring uninterrupted data processing.

**Task**: You are required to create a Python program that takes user input for source code and implements a buffering system with two buffers using multithreading.

**Requirements:**

* Initialization: Initialize two buffers, each capable of storing a fixed number of characters (e.g., 100 characters). Let's call them Buffer A and Buffer B.
* Data Processing : Separate the “words” from the input strings.
* Create two threads: one for filling the active buffer and one for processing data from the other buffer. Use a shared data structure (e.g., a queue or a shared buffer) to synchronize data between the two threads.
* Filling the Active Buffer (Producer Thread): The producer thread is responsible for reading data from the user input and storing it in the active buffer. Implement a function fill\_buffer(buffer) that reads characters from the user input and fills the specified buffer until it's full. Ensure that the producer thread continues to fill the active buffer while the consumer thread is processing data.
* Processing Data from the Inactive Buffer (Consumer Thread): The consumer thread should continuously process data from the inactive buffer. Implement a function process\_data(buffer) that reads and processes data character by character from the specified buffer. The consumer thread should not block the producer thread from filling the active buffer.
* Thread Synchronization: Use appropriate synchronization mechanisms (e.g., locks, semaphores, or condition variables) to ensure that the threads work together seamlessly and safely.
* User Interaction: Allow the user to input source code interactively. Provide an option for the user to exit the program gracefully (e.g., by typing 'q').
* Continuous Operation: The program should run continuously, allowing users to input source code while processing data from the buffers concurrently.

**Submission**: Submit your Python program along with comments explaining your code, and include a brief report discussing the advantages of using multithreading in this input buffering scheme.

**Grading**: You will be evaluated on the correctness and efficiency of your multithreaded input buffering implementation, your use of synchronization mechanisms, the clarity of your code, and any optimizations you have applied.