Walchand College of Engineering, Sangli  
Department of Computer Science and Engineering

Class: Final Year (CSE) Year: 2025-26 Semester: 1

Course: High Performance Computing Lab

**Practical No. 4**

**Exam Seat No: 22510050**

**Name : Prathmesh M. Sarwade**

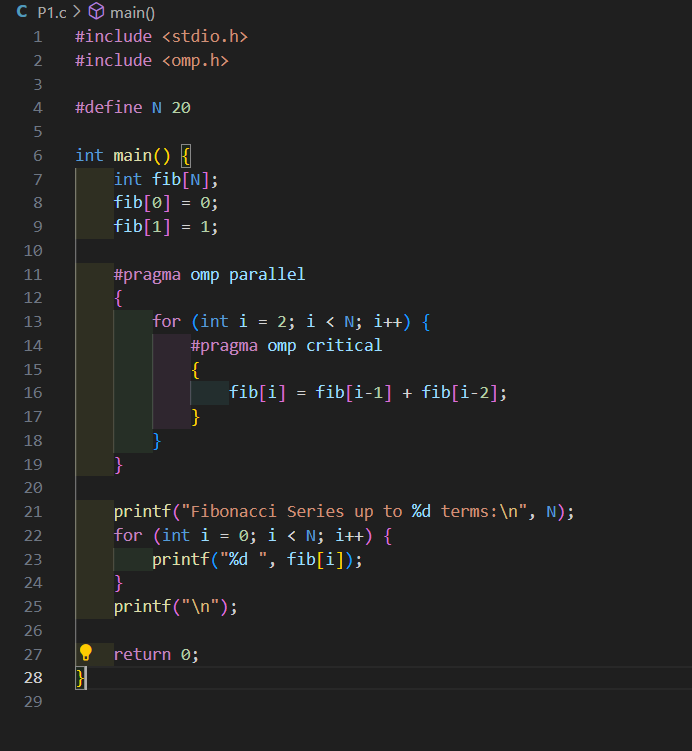
**Batch : B6**

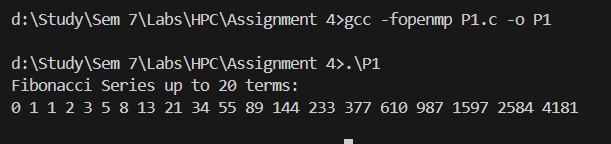
**Title of practical:**

Study and Implementation of Synchronization

**Problem Statement 1:**

# Fibonacci Computation:





The idea is to calculate Fibonacci numbers in parallel using OpenMP. While doing this, we also need to handle synchronization so that threads don’t interfere with each other. It also helps us understand how OpenMP clauses like critical, single, task, and taskwait are used.

Problem Description

Fibonacci is a sequence where each number is the sum of the two before it.

It starts with 0 and 1.

Formula:

F(0) = 0

F(1) = 1

F(n) = F(n-1) + F(n-2) for n ≥ 2

Example:

0, 1, 1, 2, 3, 5, 8, 13, 21 …

Parallelization Goal:

We want to make Fibonacci faster by running it on multiple threads.

In the iterative version, values are built step by step, which is more dependent, but we can still use OpenMP with synchronization to avoid mistakes.

Why Synchronization is Needed

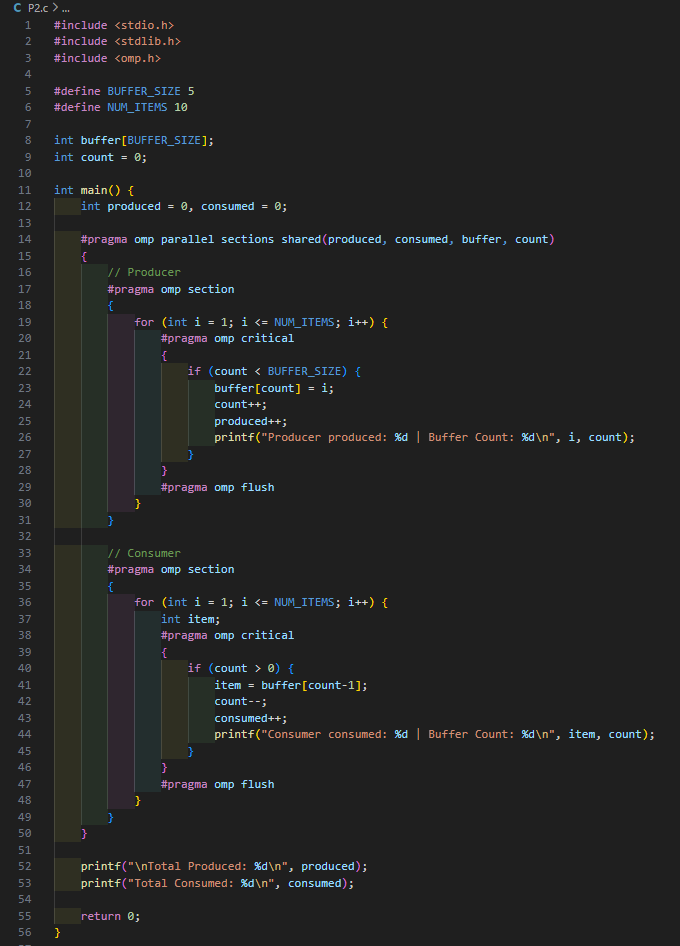
In the loop version, each Fibonacci number depends on the previous two. If two threads update or read shared data at the same time, things can go wrong (data race). That’s why we need synchronization.

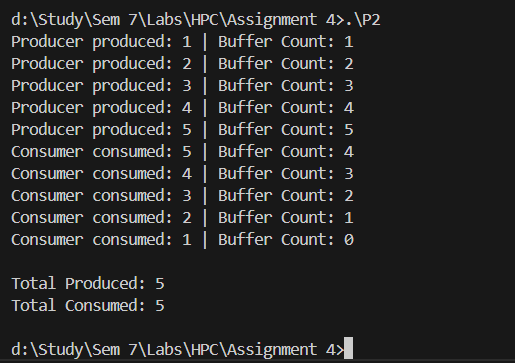
OpenMP gives us tools for this:

critical → makes sure only one thread updates at a time.

**Problem Statement 2:**

Producer Consumer Problem





**Information:**

Producer–Consumer Problem

The Producer–Consumer problem is a well-known example in concurrent programming that deals with synchronization.

The producer creates items and puts them into a common buffer.

The consumer takes items from the buffer and uses them.

Proper coordination is needed to avoid issues like:

* Buffer overflow (producer tries to add when buffer is full)
* Buffer underflow (consumer tries to remove when buffer is empty)
* Data races (multiple threads accessing/modifying shared data unsafely)

Using OpenMP for Parallelization:

We use OpenMP to run the producer and consumer at the same time, with each in its own section.

The critical makes sure only one thread changes the buffer or count at a time, preventing errors.

The flush helps keep memory updates visible to both threads, so they work correctly together.