High Performance Computing Lab : 04

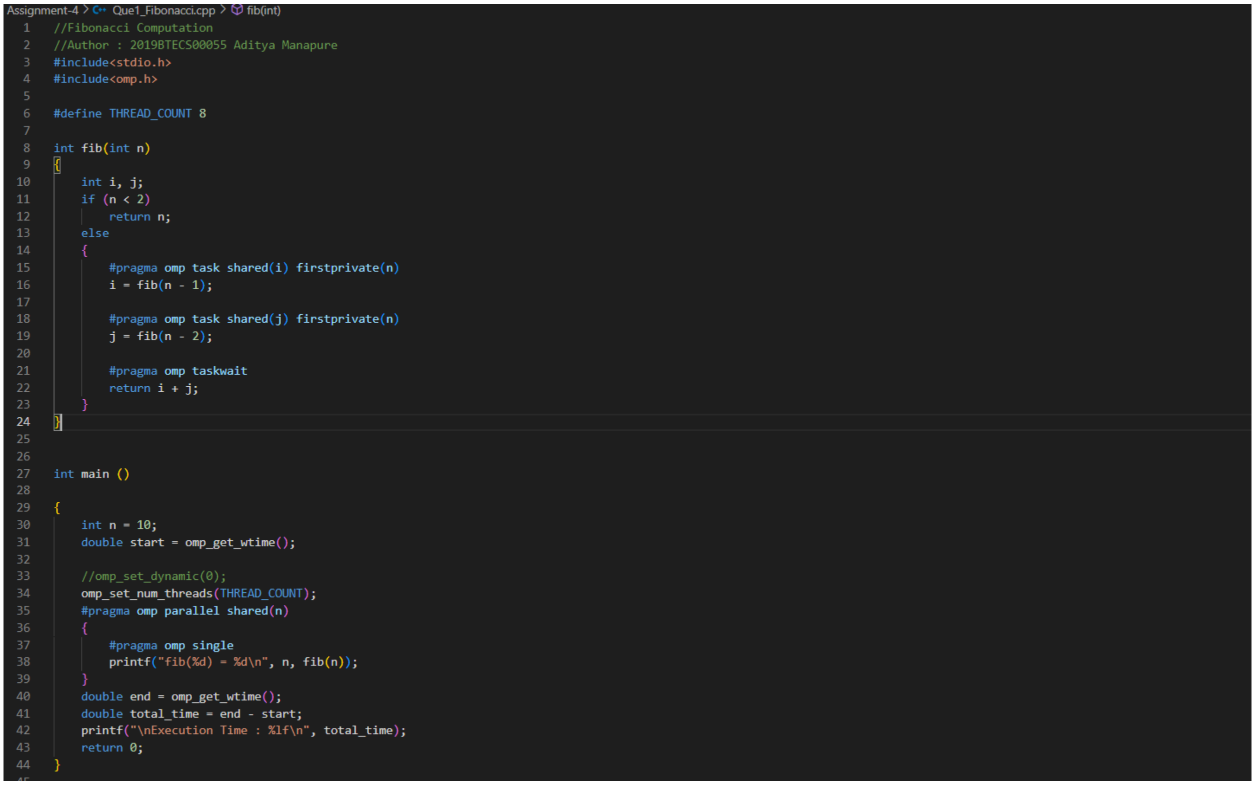
Study and Implementation of Synchronization constructs

PRN : 2019BTECS00055
Name : Aditya Rajendra Manapure

Problem Statement 1:

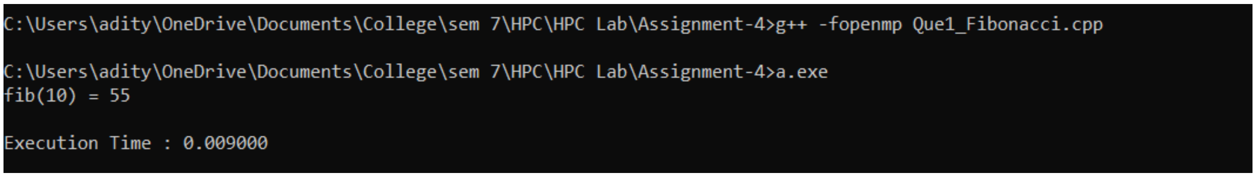
To analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements.

Screenshot 1:



```
//Fibonacci Computation
//Author : 2019BTECS00055 Aditya Manapure
#include<stdio.h>
#include<omp.h>

#define THREAD_COUNT 8

int fib(int n)
{
    int i, j;
    if (n < 2)
        return n;
    else
    {
        #pragma omp task shared(i) firstprivate(n)
        i = fib(n - 1);

        #pragma omp task shared(j) firstprivate(n)
        j = fib(n - 2);

        #pragma omp taskwait
        return i + j;
    }
}


int main ()

{
    int n = 10;
    double start = omp_get_wtime();

    //omp_set_dynamic(0);
    omp_set_num_threads(THREAD_COUNT);
    #pragma omp parallel shared(n)
    {
        #pragma omp single
        printf("fib(%d) = %d\n", n, fib(n));
    }
    double end = omp_get_wtime();
    double total_time = end - start;
    printf("\nExecution Time : %lf\n", total_time);
    return 0;
}
```

Screenshot 2:



```
C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-4>g++ -fopenmp Que1_Fibonacci.cpp

C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-4>a.exe
fib(10) = 55

Execution Time : 0.009000
```

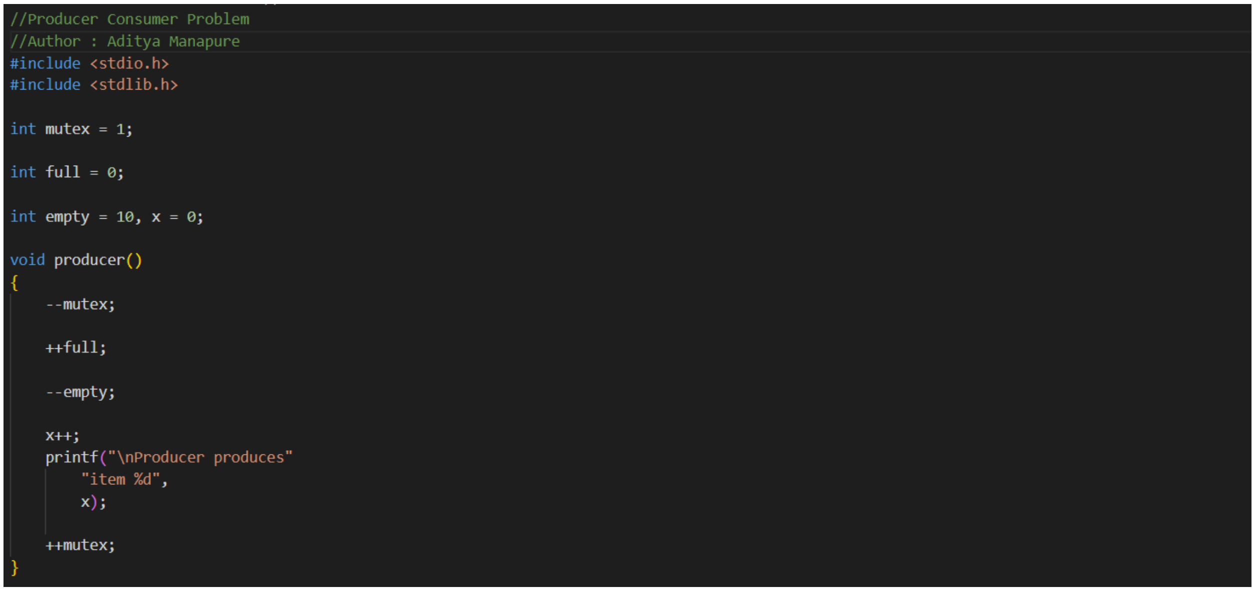
Information 1:

The shared clause declares the variables in the list to be shared among all the threads in a team. All threads within a team access the same storage area for shared variables.The firstprivate clause provides a superset of the functionality provided by the private clause. The private variable is initialized by the original value of the variable when the parallel construct is encountered.

Problem Statement 2:

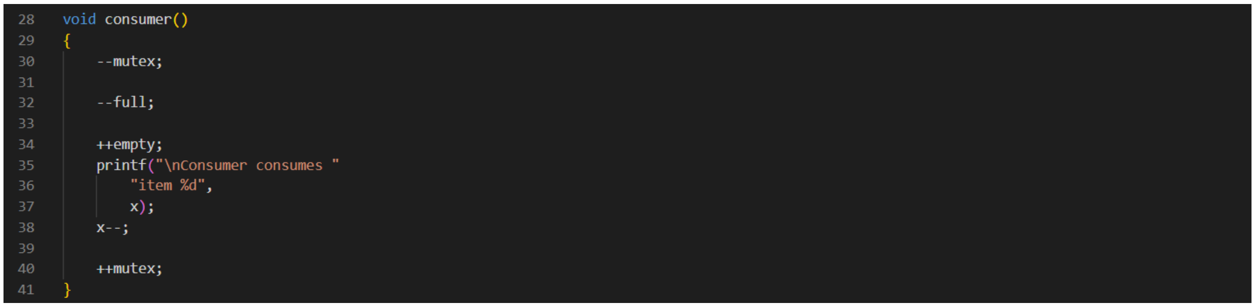
Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements.

Screenshot 1:



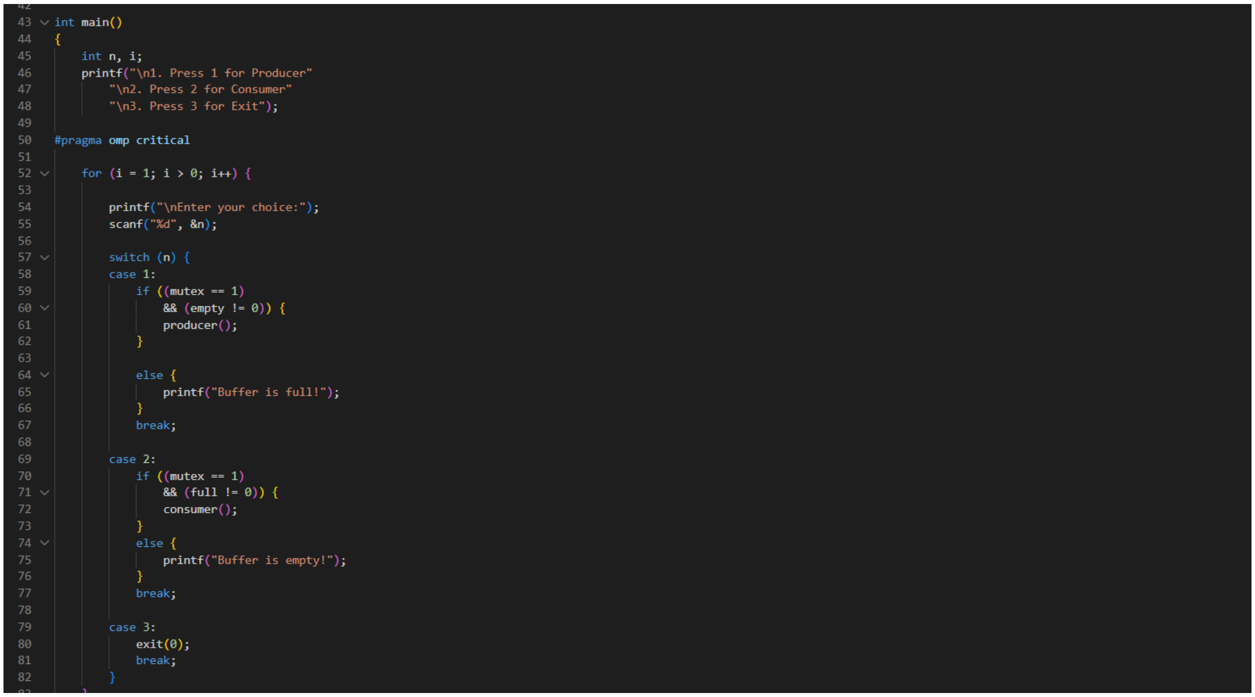
```
//Producer Consumer Problem
//Author : Aditya Manapure
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()
{
    --mutex;

    ++full;

    --empty;

    x++;
    printf("\nProducer produces"
        "item %d",
        x);

    ++mutex;
}
```

Screenshot 2:



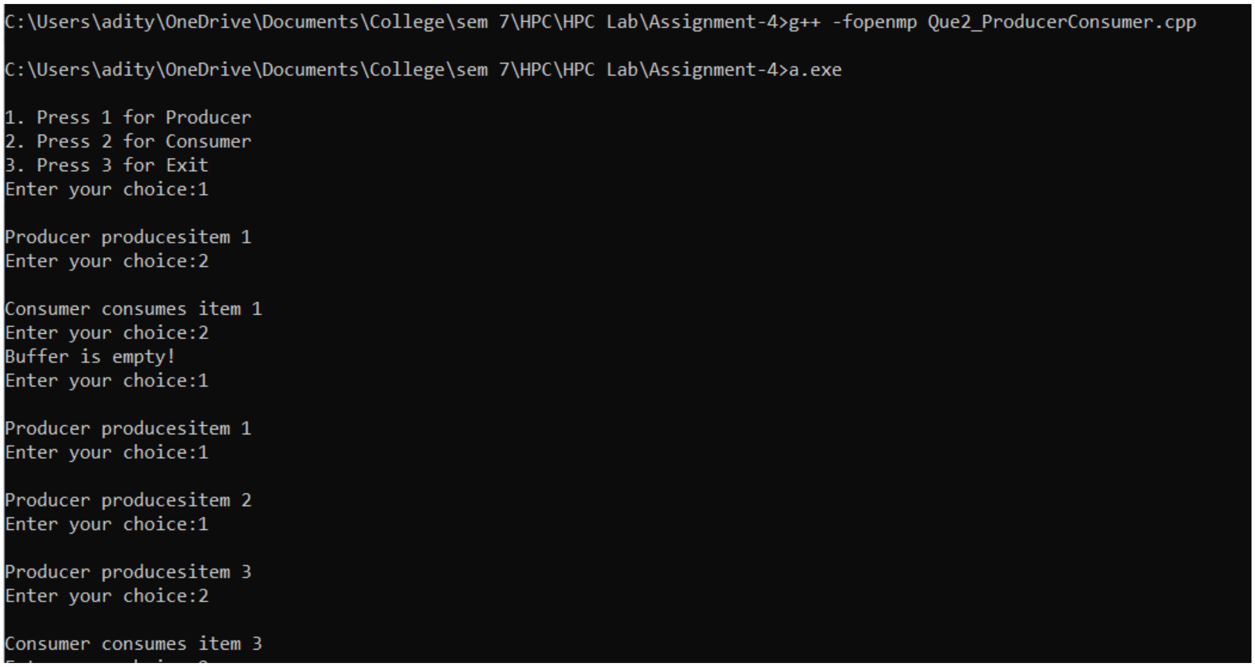
```
void consumer()
{
    --mutex;

    --full;

    ++empty;
    printf("\nConsumer consumes "
        "item %d",
        x);
    x--;

    ++mutex;
}
```

Screenshot 3:



```
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
        "\n2. Press 2 for Consumer"
        "\n3. Press 3 for Exit");

#pragma omp critical

    for (i = 1; i > 0; i++) {

        printf("\nEnter your choice:");
        scanf("%d", &n);

        switch (n) {
        case 1:
            if ((mutex == 1)
                && (empty != 0)) {
                producer();
            }

            else {
                printf("Buffer is full!");
            }
            break;

        case 2:
            if ((mutex == 1)
                && (full != 0)) {
                consumer();
            }
            else {
                printf("Buffer is empty!");
            }
            break;

        case 3:
            exit(0);
            break;
        }
    }
```

Output :



```
C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-4>g++ -fopenmp Que2_ProducerConsumer.cpp

C:\Users\adity\OneDrive\Documents\College\sem 7\HPC\HPC Lab\Assignment-4>a.exe

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer producesitem 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer producesitem 1
Enter your choice:1

Producer producesitem 2
Enter your choice:1

Producer producesitem 3
Enter your choice:2

Consumer consumes item 3
```

Information 2:

A thread waits at the start of a critical region identified by a given name until no other thread in the program is executing a critical region with that same name. Critical sections not specifically named by omp critical directive invocation are mapped to the same unspecified name.