**Name: Varun Menon** **Course Code: CSE4001**

**Reg No: 19BCE1438** **Faculty: Dr. Harini S**

---

---

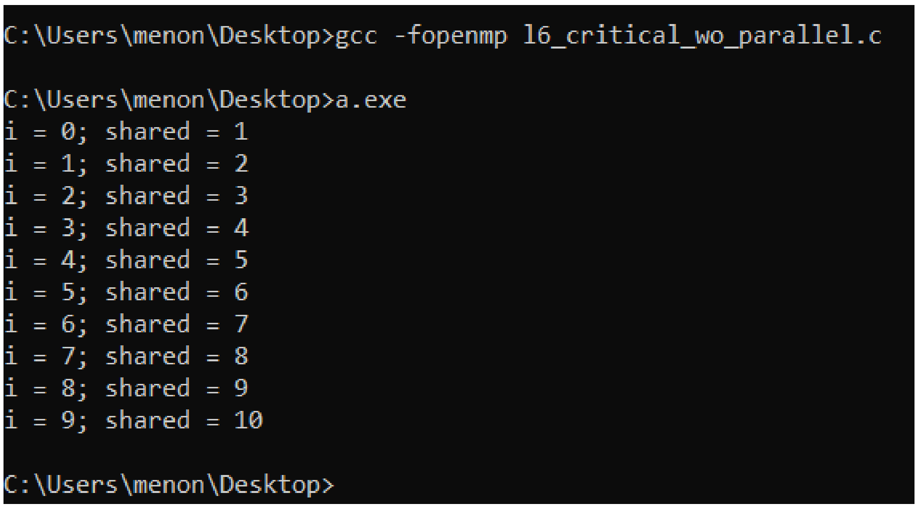
**Lab Experiment 6**

1. Sample critical without parallel program

   Code:

```c
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include<math.h>

int main()
{
    int shared = 0;

    for(int i = 0; i < 10; i++)
    {
        #pragma omp critical
            shared++;
        printf("i = %d; shared = %d\n", i, shared);
    }
    return 0;
}
```

   Output:



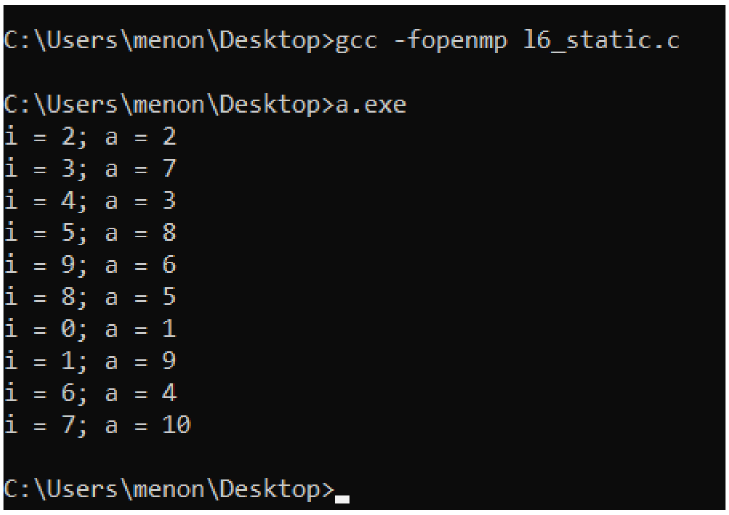
```
C:\Users\menon\Desktop>gcc -fopenmp l6_critical_wo_parallel.c

C:\Users\menon\Desktop>a.exe
i = 0; shared = 1
i = 1; shared = 2
i = 2; shared = 3
i = 3; shared = 4
i = 4; shared = 5
i = 5; shared = 6
i = 6; shared = 7
i = 7; shared = 8
i = 8; shared = 9
i = 9; shared = 10

C:\Users\menon\Desktop>
```

2. Sample static schedule program

Code:

```c
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include<math.h>

int main()
{
    int a = 0;
    int i;
    #pragma omp parallel for schedule(static)
    for(i = 0; i < 10; i++) {
        a++;
        printf("i = %d; a = %d\n", i, a);
    }
    return 0;
}
```

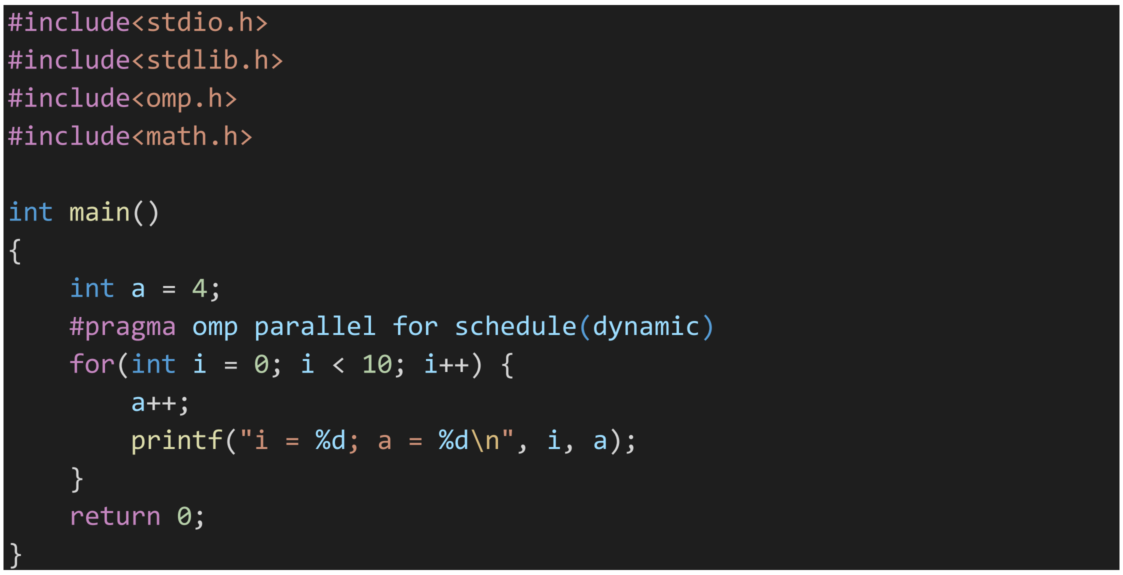
Output:



```
C:\Users\menon\Desktop>gcc -fopenmp 16_static.c

C:\Users\menon\Desktop>a.exe
i = 2; a = 2
i = 3; a = 7
i = 4; a = 3
i = 5; a = 8
i = 9; a = 6
i = 8; a = 5
i = 0; a = 1
i = 1; a = 9
i = 6; a = 4
i = 7; a = 10

C:\Users\menon\Desktop>_
```

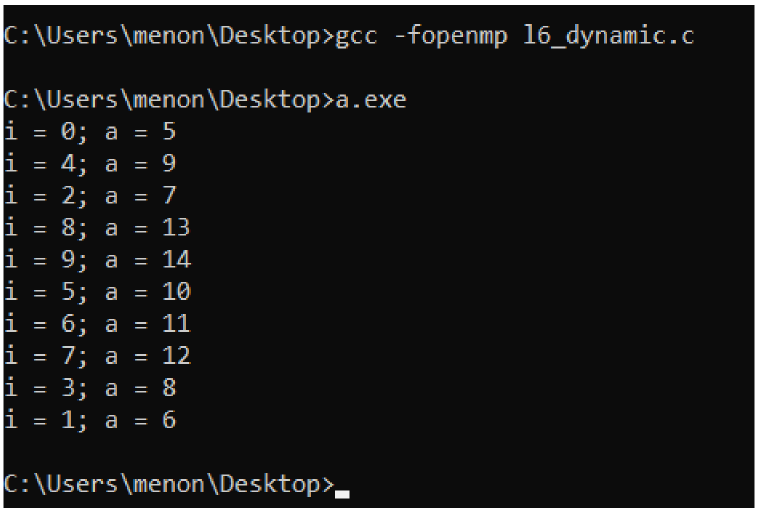
3. Sample dynamic schedule program

Code:



```c
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include<math.h>

int main()
{
    int a = 4;
    #pragma omp parallel for schedule(dynamic)
    for(int i = 0; i < 10; i++) {
        a++;
        printf("i = %d; a = %d\n", i, a);
    }
    return 0;
}
```

Output:



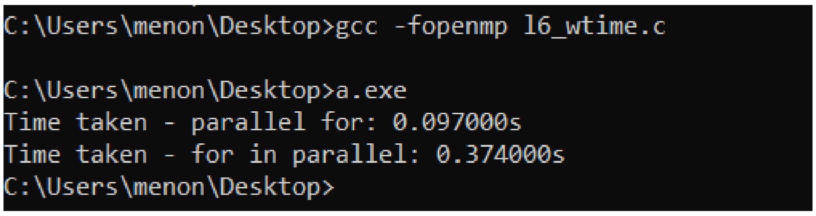
```
C:\Users\menon\Desktop>gcc -fopenmp 16_dynamic.c

C:\Users\menon\Desktop>a.exe
i = 0; a = 5
i = 4; a = 9
i = 2; a = 7
i = 8; a = 13
i = 9; a = 14
i = 5; a = 10
i = 6; a = 11
i = 7; a = 12
i = 3; a = 8
i = 1; a = 6

C:\Users\menon\Desktop>_
```

4. Sample profile program with omp_get_wtime()

Code:

```c
#include <omp.h>
#include <stdio.h>

int main()
{
    double start = omp_get_wtime();

    int sum = 0;
    #pragma omp parallel for
    for (int i = 0; i < 10000000; i++)
    {
        sum += i;
    }

    double end = omp_get_wtime();
    printf("Time taken - parallel for: %fs\n", end-start);

    sum = 0;
    double start2 = omp_get_wtime();

    #pragma omp parallel
    {
        for (int i = 0; i < 10000000; i++)
        {
            sum += i;
        }
    }

    double end2 = omp_get_wtime();
    printf("Time taken - for in parallel: %fs", end2-start2);
}
```

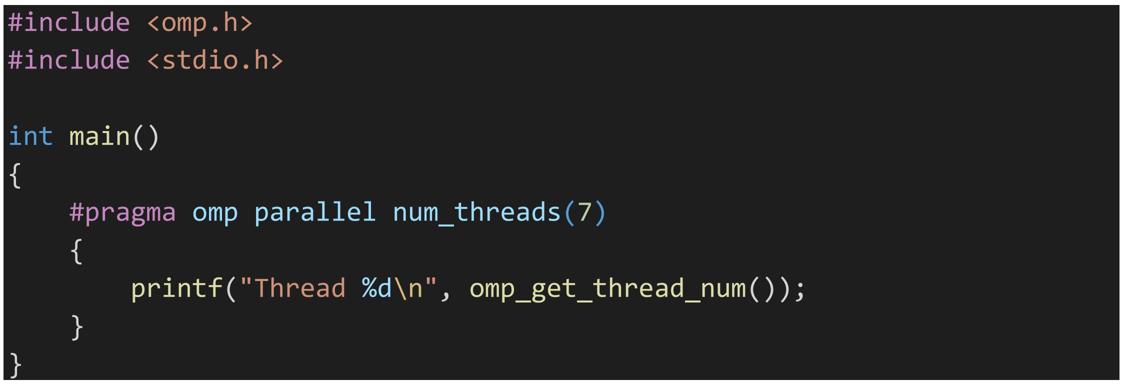
Output:



```
C:\Users\menon\Desktop>gcc -fopenmp l6_wtime.c

C:\Users\menon\Desktop>a.exe
Time taken - parallel for: 0.097000s
Time taken - for in parallel: 0.374000s
C:\Users\menon\Desktop>
```

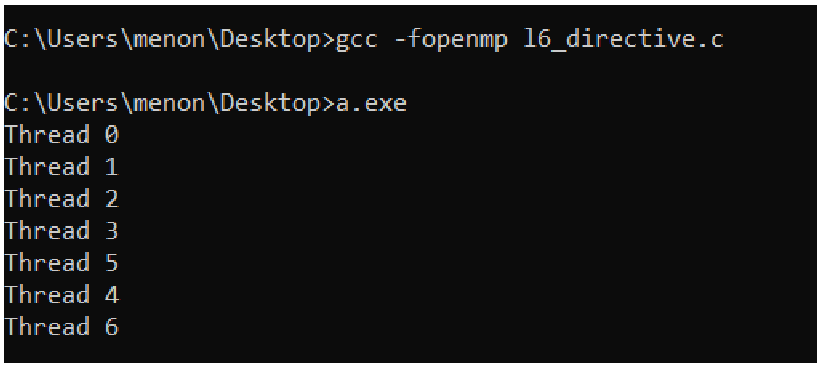
5. Sample program to set threads using directive clause

Code:



```c
#include <omp.h>
#include <stdio.h>

int main()
{
    #pragma omp parallel num_threads(7)
    {
        printf("Thread %d\n", omp_get_thread_num());
    }
}
```

Output:



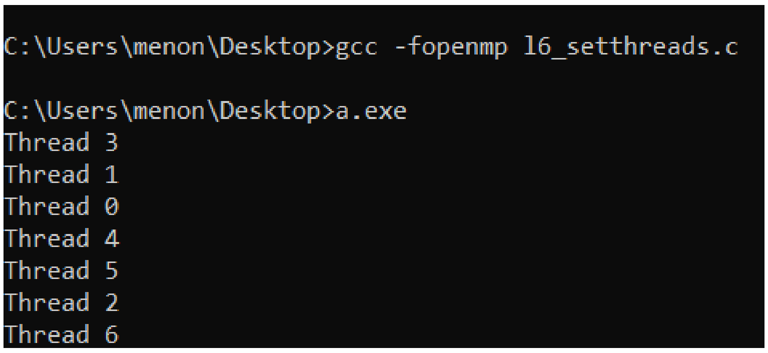
```
C:\Users\menon\Desktop>gcc -fopenmp 16_directive.c

C:\Users\menon\Desktop>a.exe
Thread 0
Thread 1
Thread 2
Thread 3
Thread 5
Thread 4
Thread 6
```

6. Sample program to set threads using omp_set_threads()

Code:

```c
#include <omp.h>
#include <stdio.h>

int main()
{
    omp_set_num_threads(7);
    #pragma omp parallel
    {
        printf("Thread %d\n", omp_get_thread_num());
    }
}
```

Output:



```
C:\Users\menon\Desktop>gcc -fopenmp 16_setthreads.c

C:\Users\menon\Desktop>a.exe
Thread 3
Thread 1
Thread 0
Thread 4
Thread 5
Thread 2
Thread 6
```

7. Optional : 3 Hot Questions
   a. Consider a priority queue that has four levels of priority. Each priority level can have at the maximum of 'n' tasks. Each task will have a priority. Depending on the priority, the tasks are inserted in the queue. For the dispatch of tasks, the tasks from the highest priority queue should be dispatched every clock cycle, from the second highest priority queue after every 2 clock cycles, from the third highest priority queue after every 3 clock cycles and so on. The insertion and dispatch can happen in parallel. Write an efficient OpenMP algorithm to emulate the above priority queue of an operating system. Every clock cycle, your program should print the following.
      i. Tasks in each priority level
      ii. Task ID last dispatched /currently getting dispatched

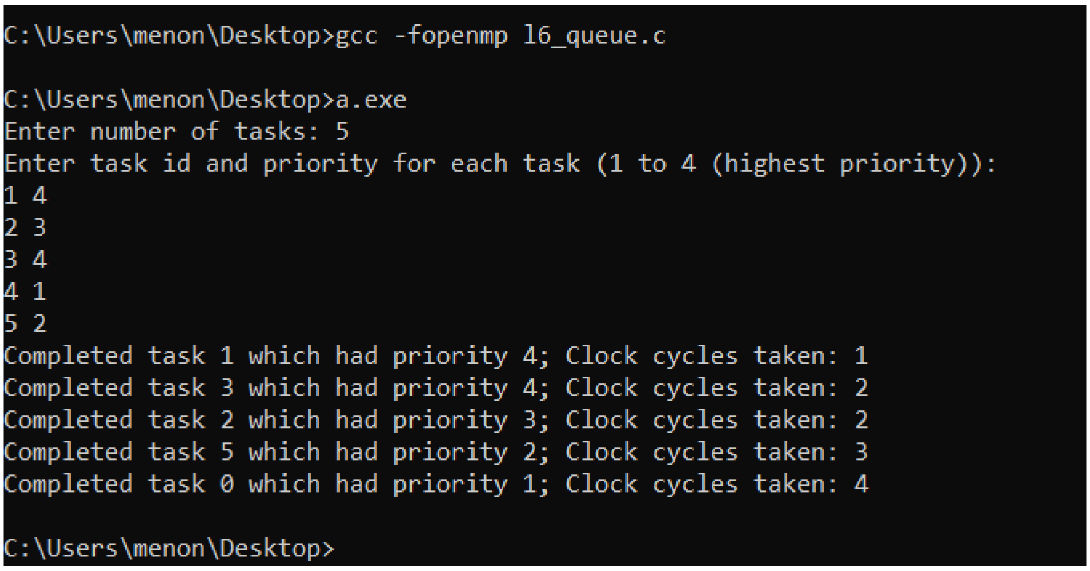
   Task ID's and its priority can be randomly generated using rand()functions. Based on the set of random inputs simulate the above scenario and display the results.

   Code:

```c
#include<omp.h>
#include<stdio.h>

int main()
{
    int n;
    printf("Enter number of tasks: ");
    scanf("%d", &n);

    int tasks[n][2];
    int front[4] = {0, 0, 0, 0}, rear[4] = {0, 0, 0, 0};
    int q[4][1000];
    printf("Enter task id and priority for each task (1 to 4
(highest priority)):\n");
    for(int i = 0; i < n; i++)
    {
        scanf("%d %d", &tasks[i][0], &tasks[i][1]);
        q[tasks[i][1]-1][rear[tasks[i][1]-1]++] = tasks[i][0]; //
enqueue to the specified queue
    }

    for(int i = 1; i < 3*n + 1; i++)
    {
        #pragma omp parallel sections
        {
            #pragma omp section
            {
                if(front[3] < rear[3])
                {
                    printf("Completed task %d which had priority
4; Clock cycles taken: %d\n", q[3][front[3]], i);
```

```
                    #pragma omp critical
                        front[3]++;
                }
            }
            #pragma omp section
            {
                if(i % 2 == 0 && front[2] < rear[2])
                {
                    printf("Completed task %d which had priority
3; Clock cycles taken: %d\n", q[2][front[2]], i);
                    #pragma omp critical
                        front[2]++;
                }
            }
            #pragma omp section
            {
                if(i % 3 == 0 && front[1] < rear[1])
                {
                    printf("Completed task %d which had priority
2; Clock cycles taken: %d\n", q[1][front[1]], i);
                    #pragma omp critical
                        front[1]++;
                }
            }
            #pragma omp section
            {
                if(i % 4 == 0 && front[0] < rear[0])
                {
                    printf("Completed task %d which had priority
1; Clock cycles taken: %d\n", q[1][front[1]], i);
                    #pragma omp critical
                        front[0]++;
                }
            }
        }
    }
    return 0;
}
```

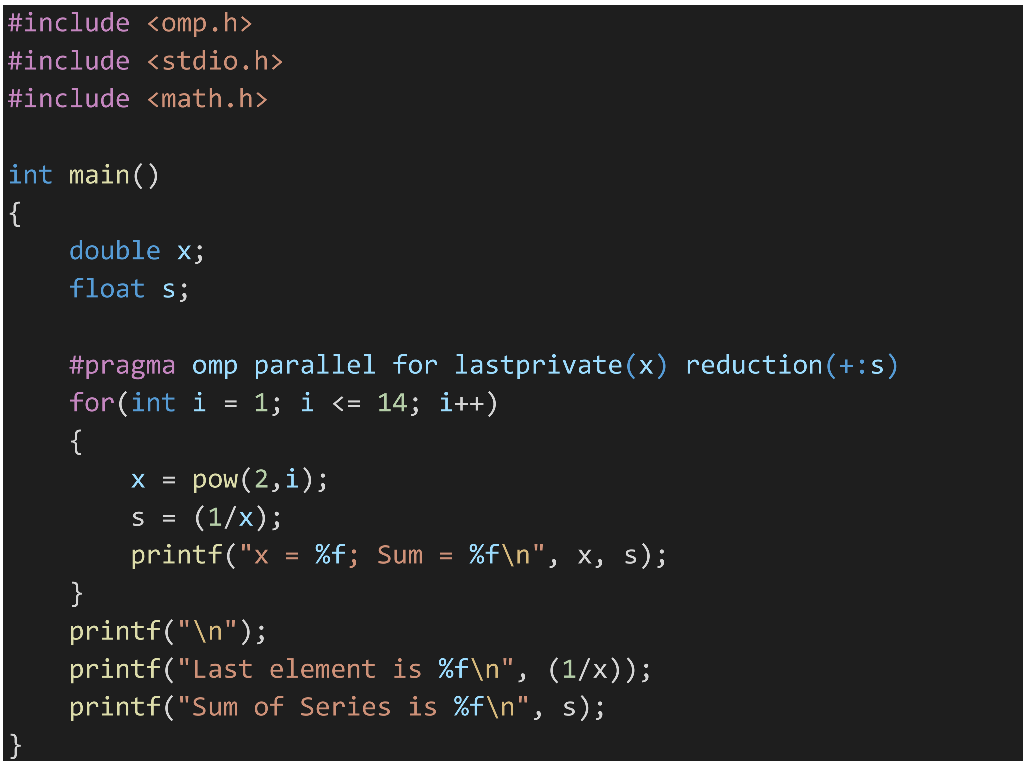
Output:



```
C:\Users\menon\Desktop>gcc -fopenmp l6_queue.c

C:\Users\menon\Desktop>a.exe
Enter number of tasks: 5
Enter task id and priority for each task (1 to 4 (highest priority)):
1 4
2 3
3 4
4 1
5 2
Completed task 1 which had priority 4; Clock cycles taken: 1
Completed task 3 which had priority 4; Clock cycles taken: 2
Completed task 2 which had priority 3; Clock cycles taken: 2
Completed task 5 which had priority 2; Clock cycles taken: 3
Completed task 0 which had priority 1; Clock cycles taken: 4

C:\Users\menon\Desktop>
```

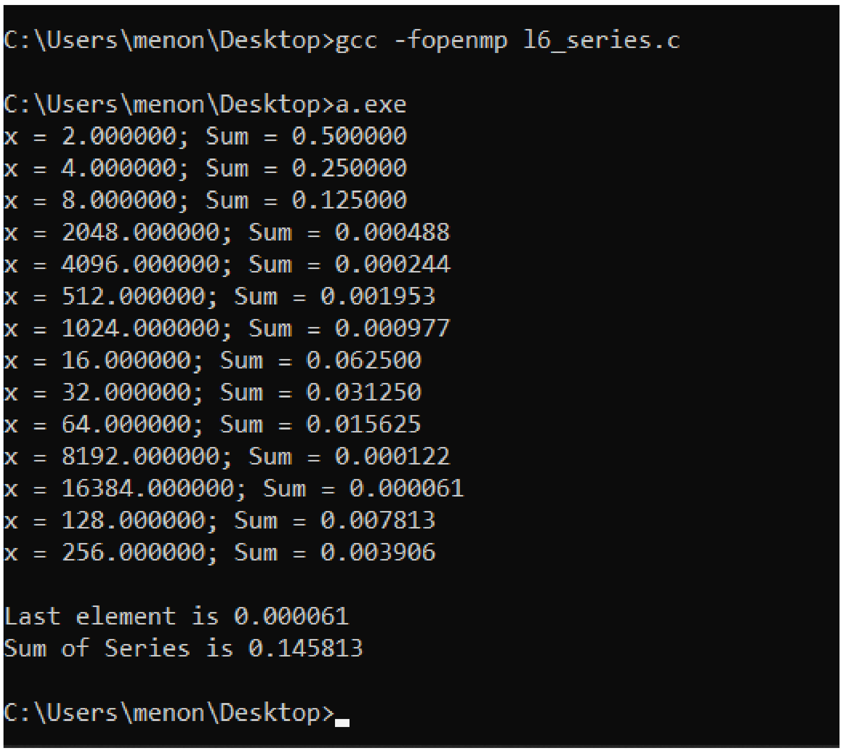
b. Write a parallel program using OpenMP to implement the following series, 1/2+1/4+1/8+1/16+……….1/16384
Find the sum of the series and print it along with the thread id and the last value in the series i.e., (1/16384).

Code:



```c
#include <omp.h>
#include <stdio.h>
#include <math.h>

int main()
{
    double x;
    float s;

    #pragma omp parallel for lastprivate(x) reduction(+:s)
    for(int i = 1; i <= 14; i++)
    {
        x = pow(2,i);
        s = (1/x);
        printf("x = %f; Sum = %f\n", x, s);
    }
    printf("\n");
    printf("Last element is %f\n", (1/x));
    printf("Sum of Series is %f\n", s);
}
```

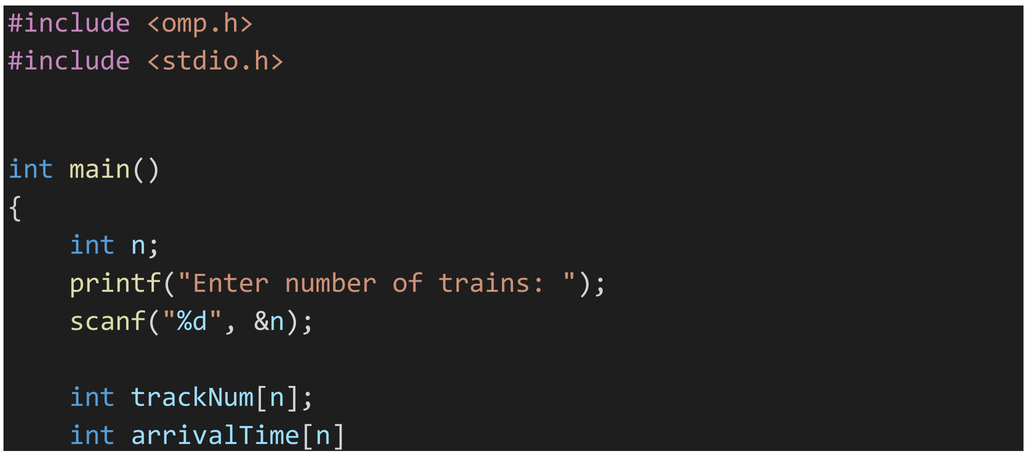
Output:



```
C:\Users\menon\Desktop>gcc -fopenmp l6_series.c

C:\Users\menon\Desktop>a.exe
x = 2.000000; Sum = 0.500000
x = 4.000000; Sum = 0.250000
x = 8.000000; Sum = 0.125000
x = 2048.000000; Sum = 0.000488
x = 4096.000000; Sum = 0.000244
x = 512.000000; Sum = 0.001953
x = 1024.000000; Sum = 0.000977
x = 16.000000; Sum = 0.062500
x = 32.000000; Sum = 0.031250
x = 64.000000; Sum = 0.015625
x = 8192.000000; Sum = 0.000122
x = 16384.000000; Sum = 0.000061
x = 128.000000; Sum = 0.007813
x = 256.000000; Sum = 0.003906

Last element is 0.000061
Sum of Series is 0.145813

C:\Users\menon\Desktop>_
```

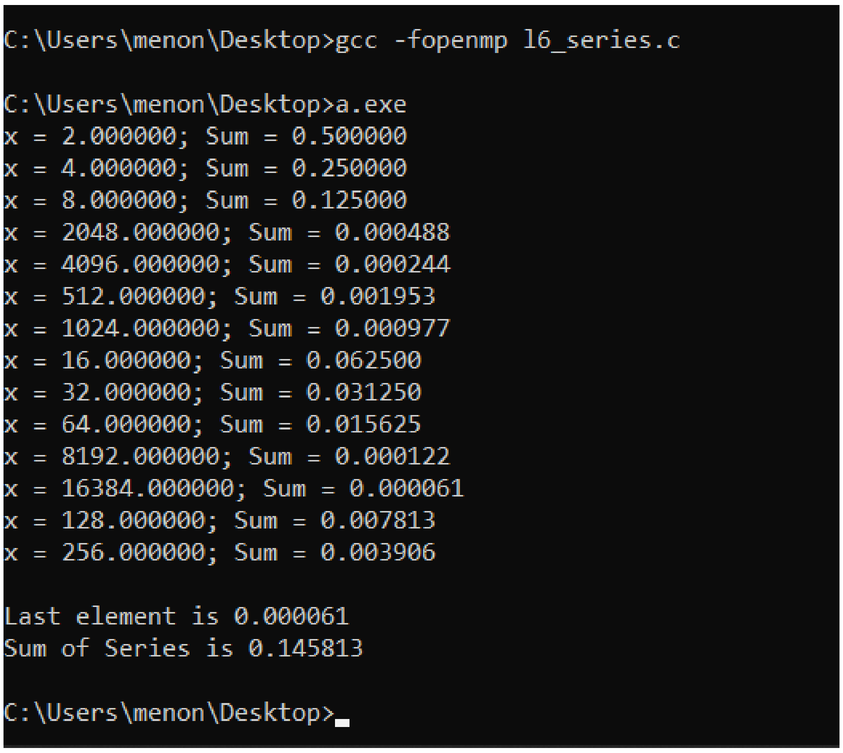
c. There are four rail tracks that pass through a railway station. The trains can use the same track with a delay of 60 minutes. Depending on the above conditions, the signal for any of the track will be updated by a control system. The signal will be displayed red for time delay less than 60. If the time delay exceeds 60, the signal will be changed to green. The above tasks are done in parallel to avoid any mishaps. Implement the above scenario in C and parallelize it using OpenMP. Let the train arrival times are randomly generated in your program and number of trains that are going to considered can be obtained from the user. The track to be used by a train can also be set by the user prior to the simulation. Whenever a signal is set, the same has to be displayed in your screen (Eg: Track 1: red, Track2 :Green etc) by your program.

Code:



```c
#include <omp.h>
#include <stdio.h>


int main()
{
    int n;
    printf("Enter number of trains: ");
    scanf("%d", &n);

    int trackNum[n];
    int arrivalTime[n]
    for (int i = 0; i < n; i++)
    {
        printf("Enter track number of train number (1 to 4) %d:",
i);
        scanf("%d", &trackNum[i]);
        arrivalTime[i] = rand();
    }

    int signals = {0,0,0,0}; // 0 is green and 1 is red
    int last_time[4] =
{omp_get_wtime(),omp_get_wtime(),omp_get_wtime(),omp_get_wtime()}
;
    omp_set_num_threads(4);

    for (int i = 0; i < n; i++)
    {
        #pragma omp parallel sections
            #pragma omp section
            {
                if (trackNum[i] == 1 && ((last_time[0] -
omp_get_wtime()) <= 60*60))
                {
                    #pragma omp critical
                    {
                        signals[0] = 1; //red
                        last_time[0] = omp_get_wtime();
                    }
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }
                }
                else if (((last_time[0] - omp_get_wtime()) <=
60*60))
                {
                    signals[0] = 0; //green
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }
                }
            }
            #pragma omp section
            {
                if (trackNum[i] == 2 && ((last_time[1] -
omp_get_wtime()) <= 60*60))
                {
                    #pragma omp critical
                    {
                        signals[1] = 1; //red
                        last_time[1] = omp_get_wtime();
                    }
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }
                }
                else if (((last_time[1] - omp_get_wtime()) <=
60*60))
                {
                    signals[1] = 0; //green
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }
                }
            }
            #pragma omp section
            {
                if (trackNum[i] == 3 && ((last_time[2] -
omp_get_wtime()) <= 60*60))
                {
                    #pragma omp critical
                    {
                        signals[2] = 1; //red
                        last_time[2] = omp_get_wtime();
                    }
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }
                }
                else if (((last_time[2] - omp_get_wtime()) <=
60*60))
                {
                    signals[2] = 0; //green
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }
                }
            }
            #pragma omp section
            {
                if (trackNum[i] == 4 && ((last_time[3] -
omp_get_wtime()) <= 60*60))
                {
                    #pragma omp critical
                    {
                        signals[3] = 1; //red
                        last_time[3] = omp_get_wtime();
                    }
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }

                }
                else if (((last_time[3] - omp_get_wtime()) <=
60*60))
                {
                    signals[3] = 0; //green
                    for(int j = 0; j < 4; j++)
                    {
                        if(signals[j] == 0)
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                        else
                        {
                            printf("Signal %d: Green\n", (j+1));
                        }
                    }
                }
            }
    }

}
```

Output:



```
C:\Users\menon\Desktop>gcc -fopenmp 16_series.c

C:\Users\menon\Desktop>a.exe
x = 2.000000; Sum = 0.500000
x = 4.000000; Sum = 0.250000
x = 8.000000; Sum = 0.125000
x = 2048.000000; Sum = 0.000488
x = 4096.000000; Sum = 0.000244
x = 512.000000; Sum = 0.001953
x = 1024.000000; Sum = 0.000977
x = 16.000000; Sum = 0.062500
x = 32.000000; Sum = 0.031250
x = 64.000000; Sum = 0.015625
x = 8192.000000; Sum = 0.000122
x = 16384.000000; Sum = 0.000061
x = 128.000000; Sum = 0.007813
x = 256.000000; Sum = 0.003906

Last element is 0.000061
Sum of Series is 0.145813

C:\Users\menon\Desktop>
```