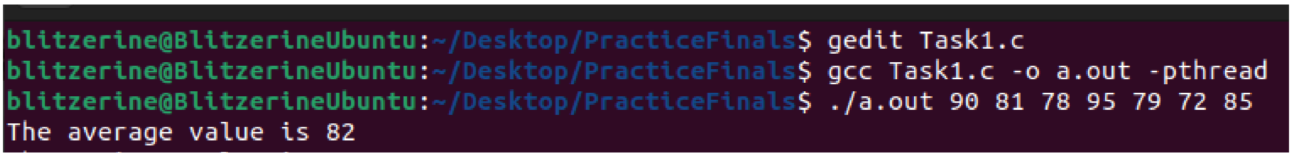
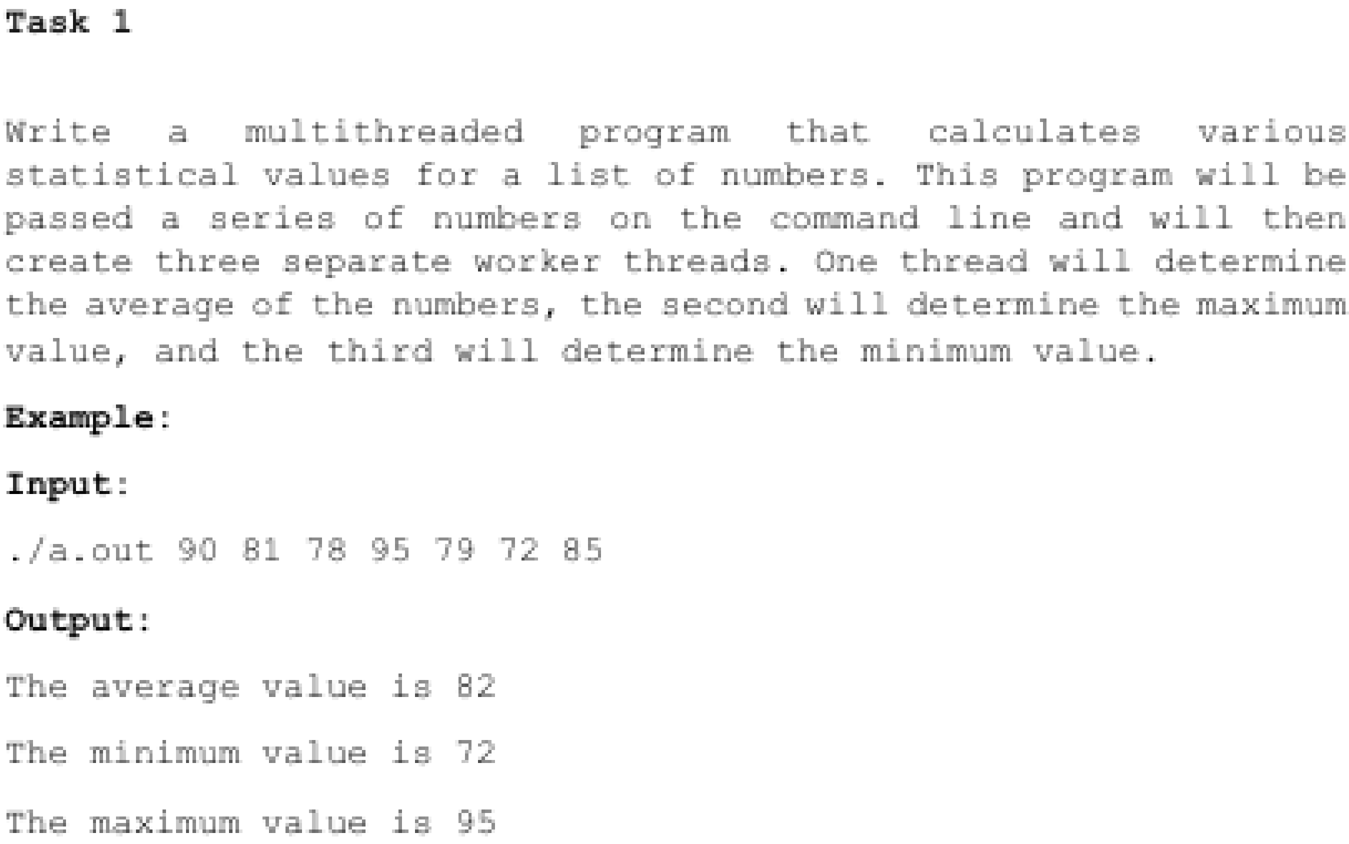
Task 1



```
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ gedit Task1.c
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ gcc Task1.c -o a.out -pthread
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./a.out 90 81 78 95 79 72 85
The average value is 82
```



Task 1

Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value.

Example:

Input:

```
./a.out 90 81 78 95 79 72 85
```

Output:

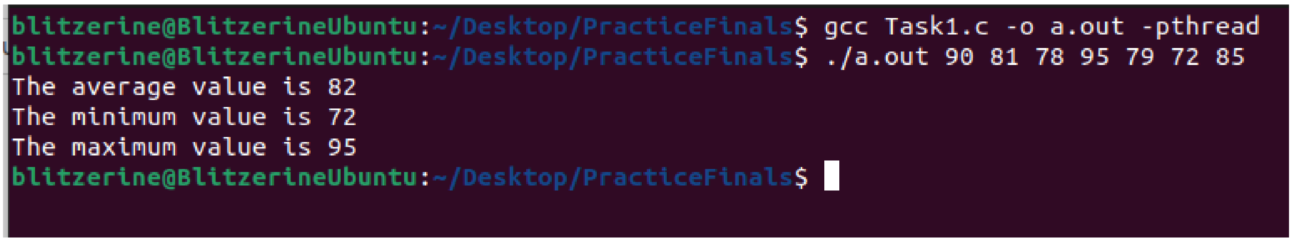
```
The average value is 82

The minimum value is 72

The maximum value is 95
```

```c
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<pthread.h>

int NumbersInArray;

void* Average(void* Arr)
{
  int Avg = 0;
  for (int i=0;i<NumbersInArray;i++)
    Avg+= *((int*)Arr +i);
   Avg = Avg/NumbersInArray;
   printf("The average value is %d\n",Avg);
}

void* Minimum(void* Arr)
{
  int Min = *((int*)Arr + 0);
  for (int i=1;i<NumbersInArray;i++)
    if (Min > *((int*)Arr + i)) Min= *((int*)Arr + i);

   printf("The minimum value is %d\n",Min);
}
```

```
void* Maximum(void* Arr)
{
  int Max = *((int*)Arr + 0);
  for (int i=1;i<NumbersInArray;i++)
        if (Max < *((int*)Arr + i)) Max= *((int*)Arr + i);
   printf("The maximum value is %d\n",Max);
}

int main(int n,char* Terminal[])
{
           NumbersInArray = n-1;
           int* Array = (int*)malloc(sizeof(int) * NumbersInArray);
           pthread_t Threads[3];

           for (int i=0;i<NumbersInArray;i++)
            Array[i] = atoi(Terminal[i+1]);

             pthread_create(Threads + 0,NULL,&Average,(void*)Array);
             pthread_create(Threads + 1,NULL,&Minimum,(void*)Array);
             pthread_create(Threads + 2,NULL,&Maximum,(void*)Array);

             pthread_join(Threads[0],NULL);
             pthread_join(Threads[1],NULL);
             pthread_join(Threads[2],NULL);
    return 0;
}
```

To print them in Order
Average first then Minimum and then Maximum
do this
to Main function

```
int main(int n,char* Terminal[])
{
           NumbersInArray = n-1;
           int* Array = (int*)malloc(sizeof(int) * NumbersInArray);
           pthread_t Threads[3];

           for (int i=0;i<NumbersInArray;i++)
            Array[i] = atoi(Terminal[i+1]);

             pthread_create(Threads + 0,NULL,&Average,(void*)Array);
             pthread_join(Threads[0],NULL);
             pthread_create(Threads + 1,NULL,&Minimum,(void*)Array);
             pthread_join(Threads[1],NULL);
             pthread_create(Threads + 2,NULL,&Maximum,(void*)Array);
             pthread_join(Threads[2],NULL);
    return 0;
}
```

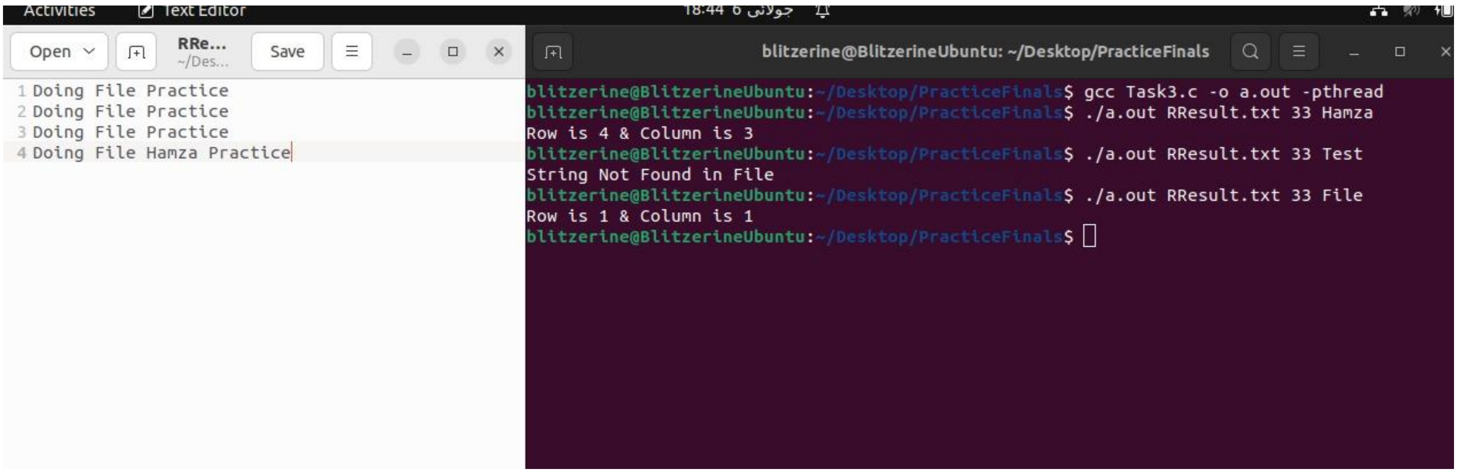


```
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ gcc Task1.c -o a.out -pthread
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./a.out 90 81 78 95 79 72 85
The average value is 82
The minimum value is 72
The maximum value is 95
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ 
```

Task 3

Task 3

Write a program that is passed a filename, a number N and a string S through command-line argument. The program opens a file to search for a string S using N number of threads. If any of the threads find sting S, it prints an appropriate message along with line and column number and exits. The other threads will also exit immediately once the string is found by any thread.



```
Doing File Practice
Doing File Practice
Doing File Practice
Doing File Hamza Practice
```

```
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ gcc Task3.c -o a.out -pthread
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./a.out RResult.txt 33 Hamza
Row is 4 & Column is 3
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./a.out RResult.txt 33 Test
String Not Found in File
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./a.out RResult.txt 33 File
Row is 1 & Column is 1
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ 
```

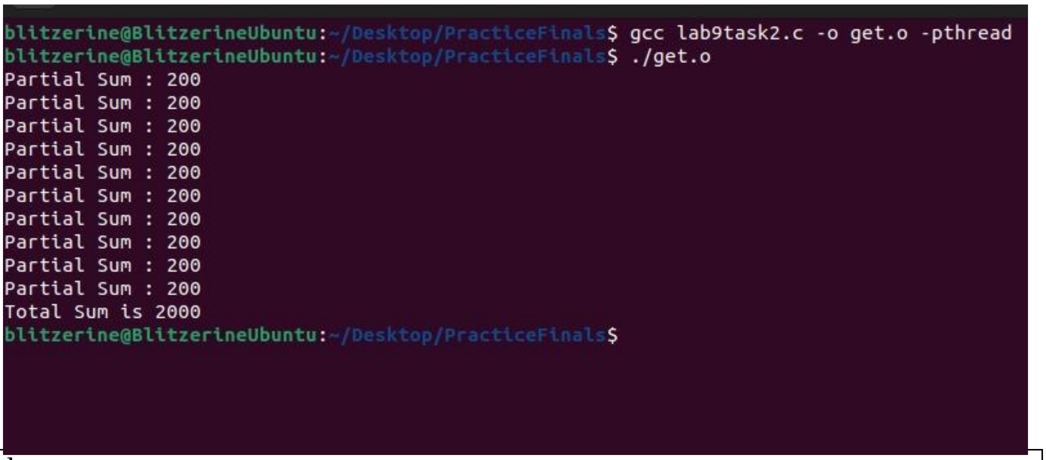
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<string.h>
#include<stdbool.h>
#include<pthread.h>

bool isFound;
char* SearchName;
char* RFileName;
int  RowFound, ColFound;

void* FindString()
{
  int fd= open(RFileName,O_RDONLY);
   char Buffer[1000];
   int index =0;
   int col = 1;
```

```
    int row = 1;
    char ch;
    int bytesRead;
    while (read(fd, &ch, 1) > 0)
     {
        if(isFound)return NULL;
        if (ch == '\n')
        {
         Buffer[index] = '\0';
         index = 0;
         if (strcmp(Buffer,SearchName)== 0)
          {
            RowFound = row;
          ColFound = col;
          isFound = true;
          return NULL;
           }
            col = 1;
            row++;
        }
        else if(ch == ' ')
        {
         Buffer[index] = '\0';
         index = 0;
         if (strcmp(Buffer,SearchName)== 0)
         {
          RowFound = row;
          ColFound = col;
          isFound = true;
          return NULL;
         }
         col++;
        }
        else Buffer[index++] = ch;
    }
  close (fd);
   return NULL;
}
int main(int n,char* Terminal[])
{

  isFound=false;
        RFileName = Terminal[1];
           SearchName = Terminal[3];
  int NoOfThreads=atoi(Terminal[2]);



  pthread_t* Threads = (pthread_t*)malloc(sizeof(pthread_t)*NoOfThreads);
  for (int i=0;i<NoOfThreads;i++)
    pthread_create(Threads+i,NULL,&FindString,NULL);

  for (int i=0;i<NoOfThreads;i++)
    pthread_join(Threads[i],NULL);

   if (!isFound) printf("String Not Found in File\n");
  else printf("Row is %d & Column is %d\n",RowFound,ColFound);

return 0;
}
```

LAB 9 Task 2

Task 2 (to be submitted in lab)

Write a program that creates ten threads to sum an array of 1000 elements. Instead of returning a partial sum from each thread, each thread should accumulate its partial sum into the shared variable 'sum', ensuring that only one thread writes to it at a time.

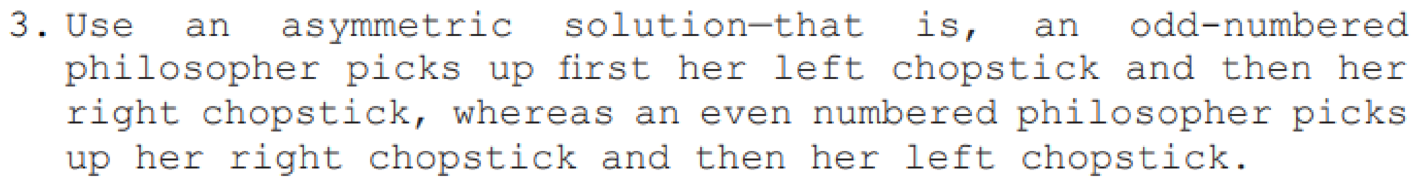


```
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ gcc lab9task2.c -o get.o -pthread
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./get.o
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Partial Sum : 200
Total Sum is 2000
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$
```

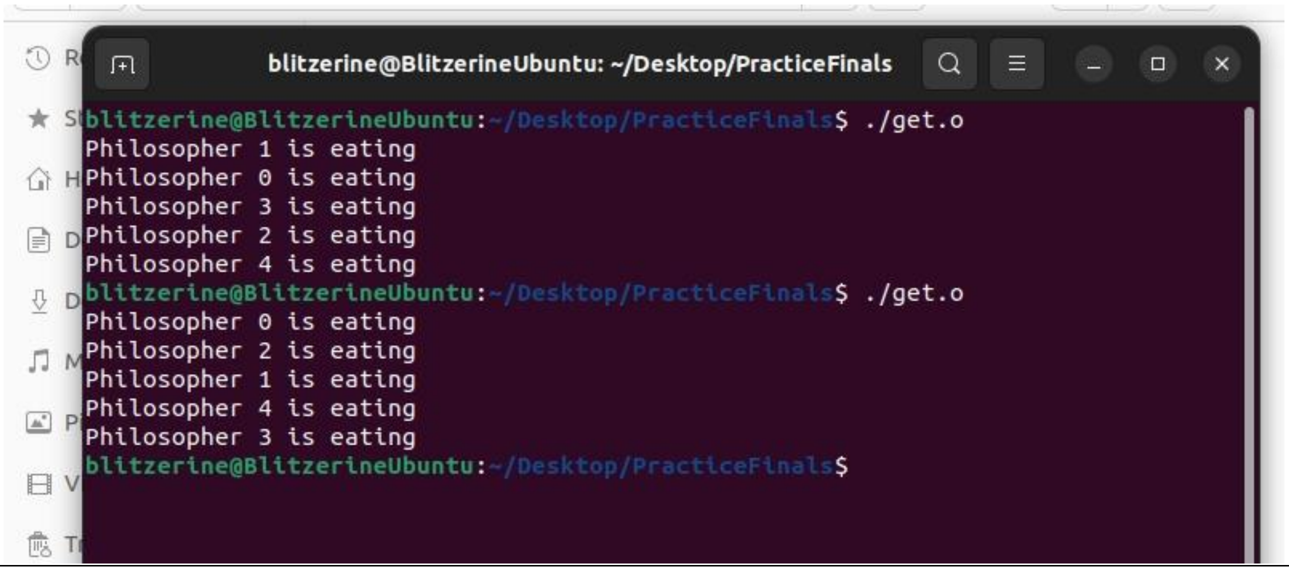
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>

int Diff;

void* Sum(void* x)
{
   int* Sum =(int*)malloc(sizeof(int));
    for (int i=0;i<Diff;i++)
       *Sum+= ((int*)x)[i];
    printf("Partial Sum : %d\n",*Sum);
    return (void*)Sum;
}
int main()
{

  int Array[1000];
  for (int i=0;i<1000;i++) Array[i] = 2;
  pthread_t Th[10];

  Diff = 1000/10;

  for (int i=0;i<10;i++)
    pthread_create(Th+i,NULL,&Sum,(void*)(Array+(i*Diff)));

  void* Getter;
  int TotalSum = 0;
  for (int i=0;i<10;i++)
  {
    pthread_join(Th[i],&Getter);
    TotalSum += (*(int*)Getter);
  }
   printf("Total Sum is %d\n",TotalSum);
 return 0;
}
```

LAB 10 TASK 1

3



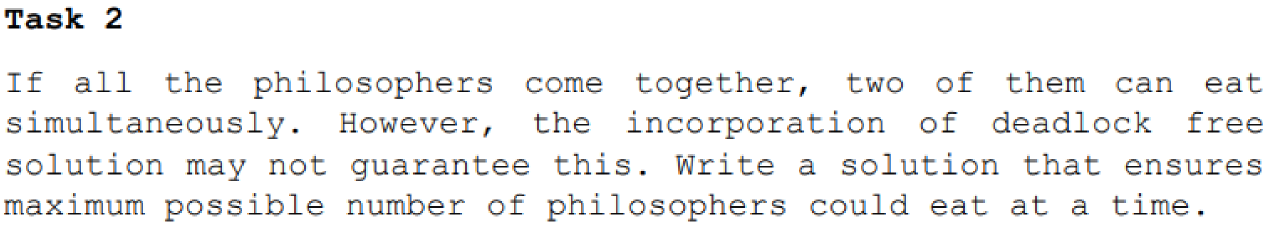
3. Use an asymmetric solution—that is, an odd-numbered philosopher picks up first her left chopstick and then her right chopstick, whereas an even numbered philosopher picks up her right chopstick and then her left chopstick.



```
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./get.o
Philosopher 1 is eating
Philosopher 0 is eating
Philosopher 3 is eating
Philosopher 2 is eating
Philosopher 4 is eating
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./get.o
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 1 is eating
Philosopher 4 is eating
Philosopher 3 is eating
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$
```

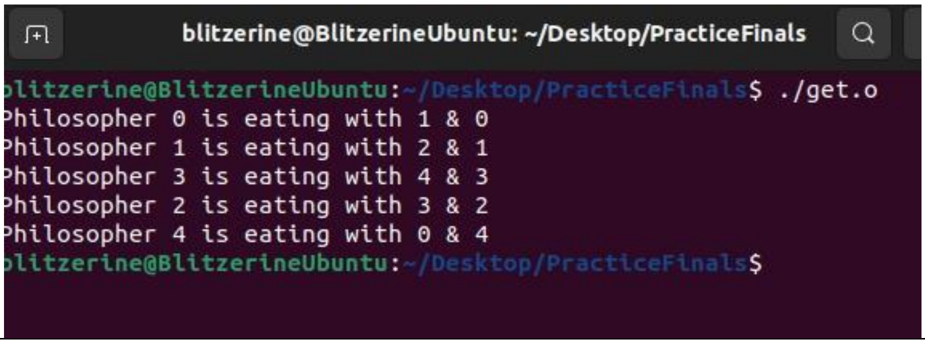
```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define N 5
sem_t chopstick[N];

void* philosopher(void* num)
{
    int i = * (int*)num;

    if (i % 2 == 0)
    {
        sem_wait(&chopstick[i]);
        sem_wait(&chopstick[(i + 1) % N]);
    }
    else
    {
        sem_wait(&chopstick[(i + 1) % N]);
        sem_wait(&chopstick[i]);
    }
    printf("Philosopher %d is eating\n", i);

    sem_post(&chopstick[i]);
    sem_post(&chopstick[(i + 1) % N]);

    return NULL;
}

int main()
 {
    pthread_t philosophers[N];
    int ids[N];

    for (int i = 0; i < N; i++)
    {
        sem_init(&chopstick[i], 0, 1);
        ids[i] = i;
    }

    for (int i = 0; i < N; i++)
        pthread_create(&philosophers[i], NULL, philosopher, &ids[i]);

    for (int i = 0; i < N; i++)
        pthread_join(philosophers[i], NULL);

    for (int i = 0; i < N; i++)
        sem_destroy(&chopstick[i]);

    return 0;
}
```

TASK 2



Task 2

If all the philosophers come together, two of them can eat simultaneously. However, the incorporation of deadlock free solution may not guarantee this. Write a solution that ensures maximum possible number of philosophers could eat at a time.



```
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./get.o
Philosopher 0 is eating with 1 & 0
Philosopher 1 is eating with 2 & 1
Philosopher 3 is eating with 4 & 3
Philosopher 2 is eating with 3 & 2
Philosopher 4 is eating with 0 & 4
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$
```

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
```

```
#include <stdlib.h>
#define N 5
sem_t chopstick[N];

void* philosopher(void* num)
{
    int i = * (int*)num;

    if (i % 2 == 0)
    {
        sem_wait(&chopstick[i]);
        sem_wait(&chopstick[(i + 1) % N]);
    }
    else
    {
        sem_wait(&chopstick[(i + 1) % N]);
        sem_wait(&chopstick[i]);
    }


    printf("Philosopher %d is eating with %d & %d \n", i, (i+1)%N , i);
    sem_post(&chopstick[i]);
    sem_post(&chopstick[(i + 1) % N]);

    return NULL;
}

int main()
 {
    pthread_t philosophers[N];
    int ids[N];

    for (int i = 0; i < N; i++)
    {
        sem_init(&chopstick[i], 0, 2);
        ids[i] = i;
    }

    for (int i = 0; i < N; i++)
        pthread_create(&philosophers[i], NULL, philosopher, &ids[i]);

    for (int i = 0; i < N; i++)
        pthread_join(philosophers[i], NULL);

    for (int i = 0; i < N; i++)
        sem_destroy(&chopstick[i]);

    return 0;
}
```

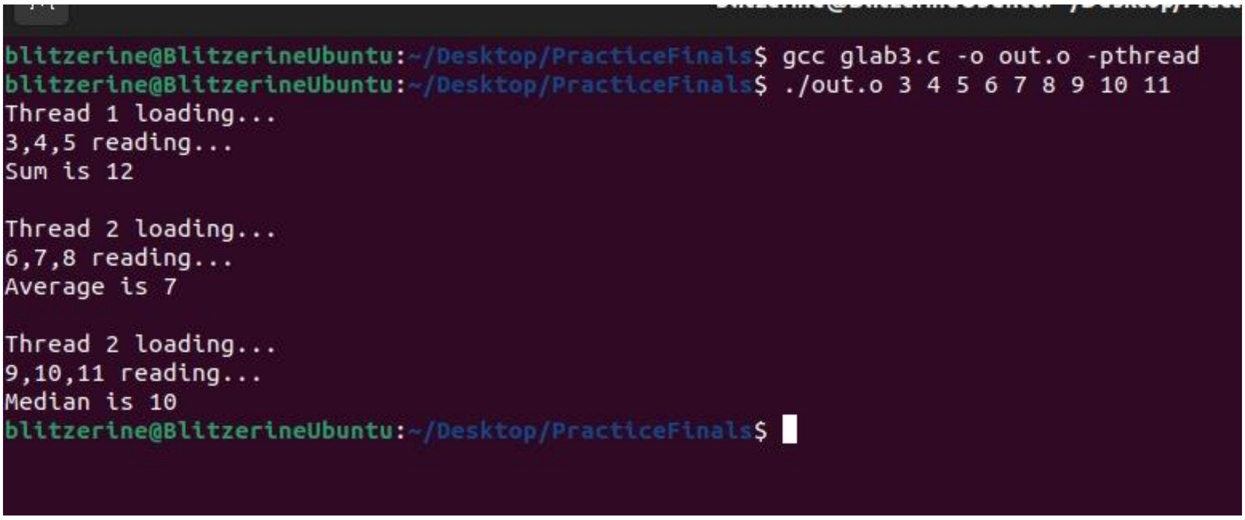
Graded Lab (Threads)

Graded Task

Write a C program in which you will take number of series as input from the command line and you will pass n/3 number to each thread. One thread will calculate sum of series. One thread will calculate average of series. One thread will calculate Median of series

Output:

```
./task1.out 3 4 5 6 7 8 9 10 11

Thread 1 Loading...

3,4,5 reading...

Sum is  12


Thread 2 Loading....

6,7,8 reading...

Average is 10.5


Thread 3 Loading....

9,10,11 reading...

Median is 10
```



```
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ gcc glab3.c -o out.o -pthread
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ ./out.o 3 4 5 6 7 8 9 10 11
Thread 1 loading...
3,4,5 reading...
Sum is 12

Thread 2 loading...
6,7,8 reading...
Average is 7

Thread 2 loading...
9,10,11 reading...
Median is 10
blitzerine@BlitzerineUbuntu:~/Desktop/PracticeFinals$ 
```

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<pthread.h>

int diff;
```

```
void* SumFun(void* Arr)
{
  int Sum = 0;
  printf("Thread 1 loading...\n");
  for (int i=0;i<diff;i++)
  {
     printf("%d", ((int*)Arr)[i]);
     if (i!=diff-1)printf(",");
     Sum+= ((int*)Arr)[i];
  }
  printf(" reading...\n");
  printf("Sum is %d\n",Sum);
}
void* AverageFun(void* Arr)
{
  int Sum = 0;
  printf("Thread 2 loading...\n");
  for (int i=0;i<diff;i++)
  {
     printf("%d", ((int*)Arr)[i]);
     if (i!=diff-1)printf(",");
     Sum+= ((int*)Arr)[i];
  }
  printf(" reading...\n");
  printf("Average is %d\n",Sum/diff);

}
void* MedianFun(void* Arr)
{
  printf("Thread 2 loading...\n");
  for (int i=0;i<diff;i++)
  {
     printf("%d", ((int*)Arr)[i]);
     if (i!=diff-1)printf(",");
  }
  printf(" reading...\n");
  printf("Median is %d\n",((int*)Arr)[diff/2]);

}
int main(int n,char* Terminal[])
{
  diff = (n-1)/3;
  pthread_t th[3];

  int* Array = (int*)malloc(sizeof(int)* (n-1));

  for (int i=0;i<n-1;i++)
     Array[i] = atoi(Terminal[i+1]);


  pthread_create(th+0,NULL,&SumFun,(void*)(Array+(0*diff)));
  pthread_join(th[0],NULL);
  printf("\n");
   pthread_create(th+1,NULL,&AverageFun,(void*)(Array+(1*diff)));
  pthread_join(th[1],NULL);
  printf("\n");
   pthread_create(th+2,NULL,&MedianFun,(void*)(Array+(2*diff)));
  pthread_join(th[2],NULL);

 return 0;
}
```