Adhesh Reghu Kumar
COE18B001

Operating Systems Lab

Lab Assignment - 5

1. Implement using threads : Armstrong number generation up to a given range.

Logic Used

In this program we are supposed to generate Armstrong numbers up to a given range n using multi-threading.
In this program, we loop from 1 to n and check if the number is an Armstrong number (sum of cubes of digits == number) in the runner function of the thread. That means we create a new thread for each number in the loop. Therefore we create n threads in the entire program. After checking and printing the result we exit the thread.

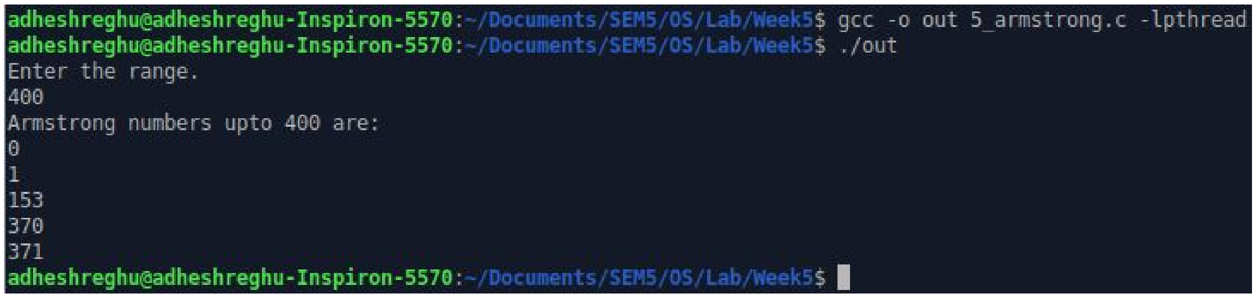
Total number of threads created = n

Code

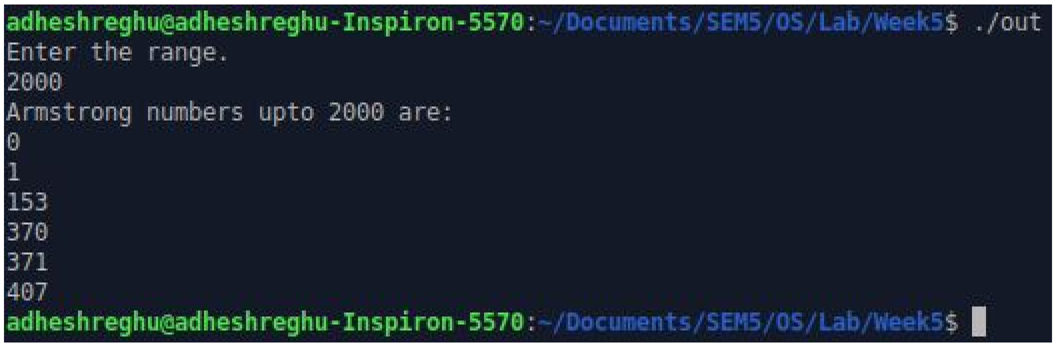
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>
#include <pthread.h>
#define MAX 1024

void* runner(void *arg);

int main()
{
    // Get range.
    int n;
    printf("Enter the range.\n");
    scanf("%d",&n);
    printf("Armstrong numbers upto %d are:\n",n);

    // Loop to range.
```

```c
    pthread_t tid[n];
    int i;
    for(i=0;i<=n;++i)
    {
       // Declare the argument to be passed to the thread
       int *arg = malloc(sizeof(*arg));
       *arg = i;

       // Create the thread
       pthread_create(&tid[i],NULL,&runner,arg);
    }

    return 0;
}

void* runner(void *arg)
{
    long int num = *((int *) arg);
    free(arg);

    // Perform the Armstrong Number Check
    long int cube_num =0, temp = num;
    int digit=0;
    while(temp!=0)
    {
       digit = temp%10;
       cube_num = cube_num + (digit * digit * digit);
       temp = temp/10;
    }

    if(cube_num == num)
       printf("%ld\n",num);

    pthread_exit(NULL);
}
```

Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 5_armstrong.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the range.
400
Armstrong numbers upto 400 are:
0
1
153
370
371
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the range.
2000
Armstrong numbers upto 2000 are:
0
1
153
370
371
407
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

2. Implement using threads : Sort an input array in ascending and descending order in multi-threaded fashion.

Logic Used

In this program we are required to sort an input array to ascending order in one thread and in descending order in another thread.
Inorder to accomplish this, we need to pass the array, then length of the array and the order of sorting to the runner() function of the thread. So we create a structure, arrayWrapper that we pass to the runner function through pthread_create() function. The arrayWrapper structure looks like this:

```
// allows the array to be passed by value (not by reference)
struct arrayWrapper
{
    int arr[MAX];
    int len;
    int order;
};
```

We first accept the input as command line arguments. Then after validating the inputs as integers using isInteger() function, we store the integer array into arr variable of an *arrayWrapper A*. We also set the len to the number of integer inputs.
Since we wish to pass the *arrayWrapper* by value, we create a copy *B* of *arrayWrapper A*. We then set the order of one structure to 0 (descending) and another structure to 1 (ascending) and pass them to different threads. The runner function sorts the *arr* of the structures according to the order given. After sorting, we exit the thread and join the main program. Once both threads have completed execution, we print the sorted array in the main program. Printing the result in the main program ensures that the order of printing is maintained.

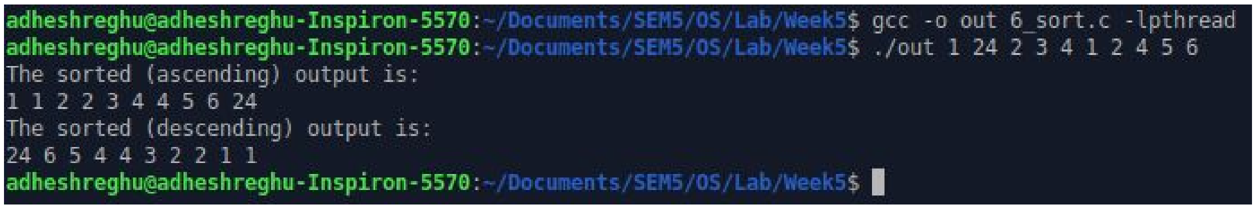
Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <pthread.h>
#define MAX 1000

int isInteger(char* string);
void sortArray(int arr[],int n,int order);

// allows the array to be passed by value (not by reference)
struct arrayWrapper
{
    int arr[MAX];
    int len;
    int order;
};

void *runner(void *arg);

int main(int argc, char *argv[])
{

    struct arrayWrapper *A = (struct arrayWrapper *)malloc(sizeof(struct
arrayWrapper));
```

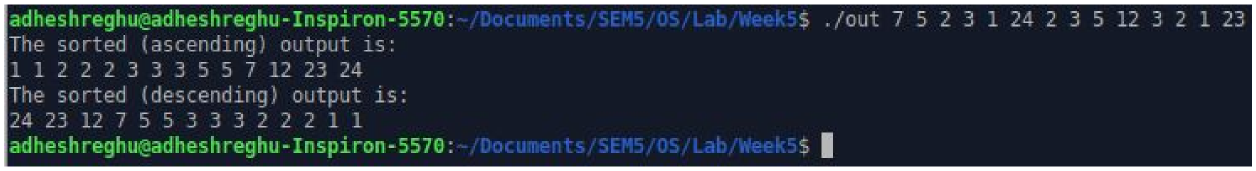
```
    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc;
       int n=0, i=1;
       for(;i<len;++i)
       {
             // Validation function. Warning if not integer.
             if(isInteger(argv[i]))
                   A->arr[n++] = atoi(argv[i]);
             else
                   printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i]);
       }
       A->len = n;

       struct arrayWrapper B = *A;

       // Warning if entire argument list is invalid.s
       if(n == 0 )
             printf("The input list is invalid. Please enter an integer
array.\n");
       else
       {
             /* -- Sort the array in two arrays in two threads
                  Create two threads -- */
             pthread_t tid[2];
             A->order = 1;
             pthread_create(&tid[0],NULL,runner,A);
             B.order = 0;
             pthread_create(&tid[1],NULL,runner,(void *)&B);

             // // Join both threads
             pthread_join(tid[0],NULL);
             pthread_join(tid[1],NULL);

             // print the results
             int i=0;
             printf("The sorted (ascending) output is:\n");
             for(;i<n;++i)
                   printf("%d ",A->arr[i]);
```

```c
             puts("");

             printf("The sorted (descending) output is:\n");
             i=0;
             for(;i<n;++i)
                   printf("%d ",B.arr[i]);
             puts("");
       }
    }
    return 0;
}


// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
       // In case of '-'' symbol, continue if not last character in string.
       if(character == 45 && i<strlen(str)-1)
             continue;
       // Else get the decimal value (-48) and check condition.
       character -= 48;
       if(character < 0 || character > 9)
             return 0;
    }
    return 1;
}


// Runner function for thread
void *runner(void *arg)
{
    int *arr = ((struct arrayWrapper*)arg)->arr;
    int order = ((struct arrayWrapper*)arg)->order;
    int n = ((struct arrayWrapper*)arg)->len;

    /* -- Sort -- */

    // For swap purposes
    int temp;
    // Initiate Loops for simple Bubble Sort.
    int i=0;
    for(;i<n;++i)
    {
       int j=0;
       for(;j<n-i-1;++j)
       {
             if(((order == 1) && (arr[j]>arr[j+1])) || ((order == 0) &&
(arr[j]<arr[j+1])))
             {
                   // Swap arr[j] and arr[j+1]
                   temp = arr[j];
                   arr[j] = arr[j+1];
                   arr[j+1] = temp;
             }
       }
    }
    pthread_exit(NULL);
}
```

Output

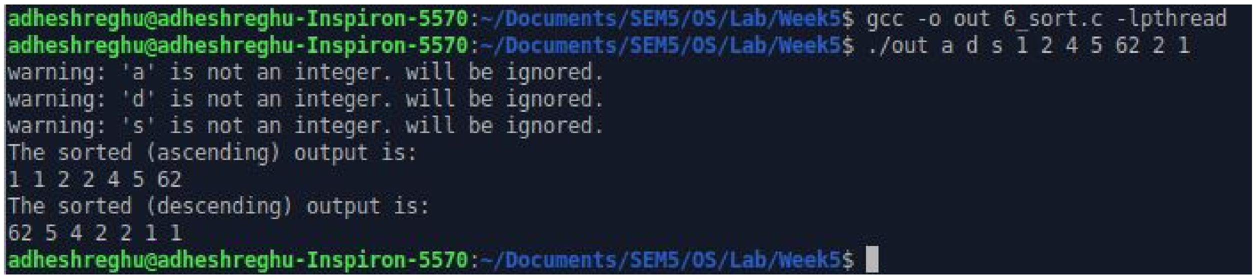


```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 6_sort.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 1 24 2 3 4 1 2 4 5 6
The sorted (ascending) output is:
1 1 2 2 3 4 4 5 6 24
The sorted (descending) output is:
24 6 5 4 4 3 2 2 1 1
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 7 5 2 3 1 24 2 3 5 12 3 2 1 23
The sorted (ascending) output is:
1 1 2 2 2 3 3 3 5 5 7 12 23 24
The sorted (descending) output is:
24 23 12 7 5 5 3 3 3 2 2 2 1 1
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: With invalid inputs



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 6_sort.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out a d s 1 2 4 5 62 2 1
warning: 'a' is not an integer. will be ignored.
warning: 'd' is not an integer. will be ignored.
warning: 's' is not an integer. will be ignored.
The sorted (ascending) output is:
1 1 2 2 4 5 62
The sorted (descending) output is:
62 5 4 2 2 1 1
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ 
```

3. Implement using threads : Binary Search.

Logic Used

In this program we are required to implement binary search in a multi-threaded fashion. One idea would be to split the entire array into 4 partitions and perform binary search in each of the partitions and increment a global counter upon finding the key *x*. Each of the four partitions could perform binary search simultaneously with the help of threads.

For this program, we maintain several global variables:

- arr[MAX] - input integer array
- arr_len - tells the length of the input array
- num_splits - tells the number of partitions the input array is divided into.
- x - the key to be searched
- count - the count of the key. Initialized to 0.

Note: For array size less than 4, the num_split = 2.

First we accept the input as command line arguments. We then validate the inputs and store them in the global array. The last integer input is taken as the key variable to be searched. Based on the num_split parameter we send the start pos of each split to a newly created thread. In the runner() of the thread, we first calculate the end. Since we assume that the array need not be sorted, we first sort each split in the thread using sortArray() function.
Once the split is sorted, we call the binarySearch() function on the split to find the position of the key. It returns the position of the key or -1 in case the key doesn't exist in the split. If the key exists in the split, we also would have to check for duplicates in

the split. For that we do a linear search on either side of the returned position.
We keep updating a local_cnt of the number of occurrences of the key in each split. At the end of the thread we update the global count and exit the thread.
In the main(), we join all the threads and print the global count.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <pthread.h>
#define MAX 1000

int isInteger(char* string);
void sortArray(int arr[],int start,int end,int order);
int binarySearch(int arr[], int start, int end, int x);

// define global array
int arr[MAX], arr_len;
// define size of each split - 4
int num_splits = 4;
// define global key
int x;
// global variable to maintain count
int count = 0;

void *runner(void *arg);

int main(int argc, char *argv[])
{
    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc-1;
       x = atoi(argv[len]);
       int n=0, i=1;
       for(;i<len;++i)
       {
```

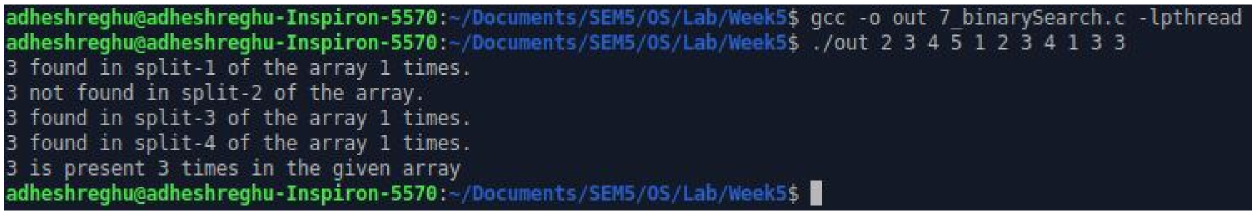
```c
             // Validation function. Warning if not integer.
             if(isInteger(argv[i]))
                   arr[n++] = atoi(argv[i]);
             else
                   printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i]);
       }
       // Assign length
       arr_len = n;

       // Warning if the entire argument list is invalid.
       if(n == 0 )
             printf("The input list is invalid. Please enter an integer
array.\n");
       else
       {
             /* -- Search for x in ? splits using Binary Search -- */
             if(arr_len < num_splits)
                   num_splits = 2;

             // Declare the threads
             pthread_t tid[num_splits];

             int split_no = 0;
             int split_size = arr_len/num_splits;

             for(i=0;i<num_splits;++i)
             {
                   int *arg = malloc(sizeof(*arg));
                   *arg = split_no;
                   pthread_create(&tid[i],NULL,runner,arg);
                   ++split_no;
             }

             for(i=0;i<num_splits;++i)
                   pthread_join(tid[i],NULL);

             printf("%d is present %d times in the given array\n",x,count);
       }
    }
    return 0;
}
```

```c
// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
       // In case of '-'' symbol, continue if not last character in string.
       if(character == 45 && i<strlen(str)-1)
             continue;
       // Else get the decimal value (-48) and check condition.
       character -= 48;
       if(character < 0 || character > 9)
             return 0;
    }
    return 1;
}

// Runner function for thread
void *runner(void *arg)
{
    int split_no = *((int *) arg);
    int start = (arr_len * split_no)/num_splits;
    int end = (arr_len * (split_no + 1))/num_splits;

    sortArray(arr,start,end,1);

    /* -- Perform Binary Search -- */
    int pos = binarySearch(arr,start,end - 1,x);
    int local_cnt = 0;
    if(pos!=-1)
    {
       ++local_cnt;
       int j = pos+1;
       if(pos+1 < end)
       {
             while((arr[j]==x) && (j<end))
             {
                   ++local_cnt;
                   j = j+1;
             }
       }
       j = pos-1;
```

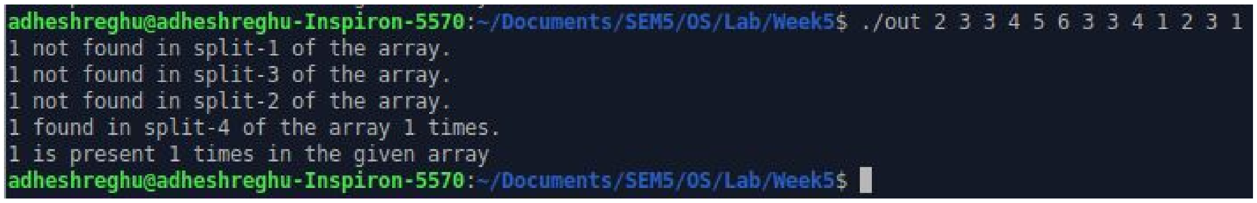
```
        if(pos-1 >= start)
        {
              while((arr[j] == x) && (j>=start))
              {
                    ++local_cnt;
                    j = j-1;
              }
        }
        printf("%d found in split-%d of the array %d
times.\n",x,split_no+1,local_cnt);
    }
    else
        printf("%d not found in split-%d of the array.\n",x,split_no+1);

    count += local_cnt;
    pthread_exit(NULL);
}


void sortArray(int arr[],int start,int end,int order)
{
    int temp;
        // For swap purposes

    // Initiate Loops for simple Bubble Sort.
    int i=start;
    int n = end - start;
    for(;i<end;++i)
    {
        int j=start;
        for(;j<start+end-i-1;++j)
        {
              if(((order == 1) && (arr[j]>arr[j+1])) || ((order == 0) &&
(arr[j]<arr[j+1])))
              {
                    // Swap arr[j] and arr[j+1]
                    temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
              }
        }
    }
}
```

```c
int binarySearch(int arr[], int start, int end, int x)
{
      if (end >= start)
      {
      int mid = start + (end - start) / 2;

      if (arr[mid] == x)
            return mid;

      if (arr[mid] > x)
            return binarySearch(arr, start, mid - 1, x);

      return binarySearch(arr, mid + 1, end, x);
      }
      return -1;
}
```

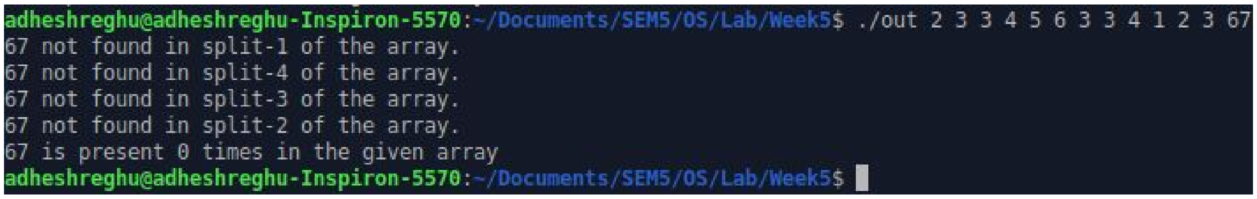
Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 7_binarySearch.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 3 4 5 1 2 3 4 1 3 3
3 found in split-1 of the array 1 times.
3 not found in split-2 of the array.
3 found in split-3 of the array 1 times.
3 found in split-4 of the array 1 times.
3 is present 3 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

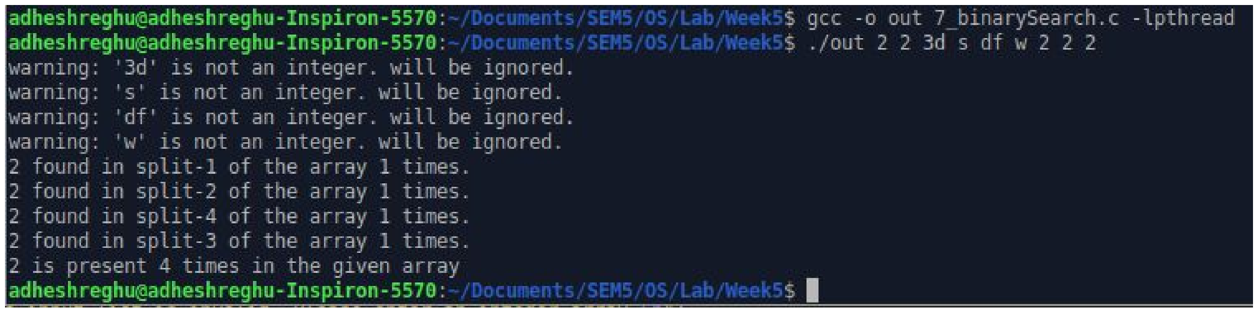


```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 3 3 4 5 6 3 3 4 1 2 3 1
1 not found in split-1 of the array.
1 not found in split-3 of the array.
1 not found in split-2 of the array.
1 found in split-4 of the array 1 times.
1 is present 1 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 3 3 4 5 6 3 3 4 1 2 3 67
67 not found in split-1 of the array.
67 not found in split-4 of the array.
67 not found in split-3 of the array.
67 not found in split-2 of the array.
67 is present 0 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: With invalid inputs



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 7_binarySearch.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 2 3d s df w 2 2 2
warning: '3d' is not an integer. will be ignored.
warning: 's' is not an integer. will be ignored.
warning: 'df' is not an integer. will be ignored.
warning: 'w' is not an integer. will be ignored.
2 found in split-1 of the array 1 times.
2 found in split-2 of the array 1 times.
2 found in split-4 of the array 1 times.
2 found in split-3 of the array 1 times.
2 is present 4 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ 
```

4. Implement using threads : Generation of prime numbers upto a limit supplied as command line parameter.

Logic Used

In this program we are required to generate the prime numbers upto a limit that is input as a command line parameter. We first take in the input n from the command line and validate it and convert it to integer.
Then we loop from 0 to n and in each iteration we create and call a thread. The thread runner takes in the loop variable i as an argument and returns the primality status of the argument. It returns 1 if the variable i is a prime number and 0 if it is not. The thread runner checks the primality status by calling a utility function isPrime().

Total Number of threads used = n

Let us take a look at the code

Code

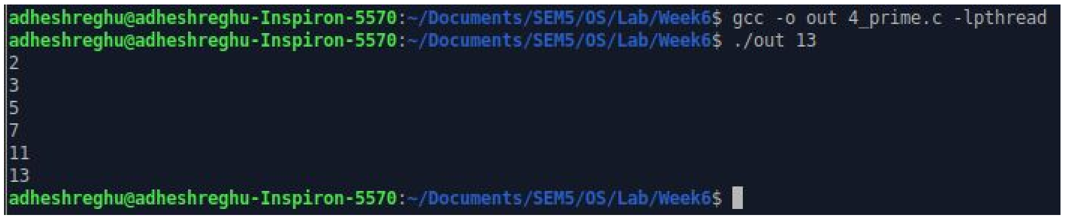
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>
#include <pthread.h>
int isInteger(char* str);
void *runner(void *arg);
```

```c
int isPrime(int n);
int main(int argc, char *argv[])
{
    // Validate the input
    if(argc <= 1)
    {
       printf("Insufficient arguments.\n");
       return 0;
    }

    // Get the input n.
    int n;
    if(isInteger(argv[1]))
       n = atoi(argv[1]);
    else
    {
       printf("Invalid input.\n");
       return 0;
    }
    // Declare n threads
    pthread_t tid[n];

    // Loop to n.
    int i=0;
    for(;i<=n;++i)
    {
       // create a thread that checks the primality of i
       int *arg = malloc(sizeof(*arg));
       *arg = i;
       pthread_create(&tid[i],NULL,runner,arg);
    }

    // Join all threads
    int *primality;
    i=0;
    for(;i<=n;++i)
    {
       // Get primality status and join thread
       pthread_join(tid[i],(void*)&primality);
       if(*primality == 1)
             printf("%d\n",i);
    }
    return 0;
```

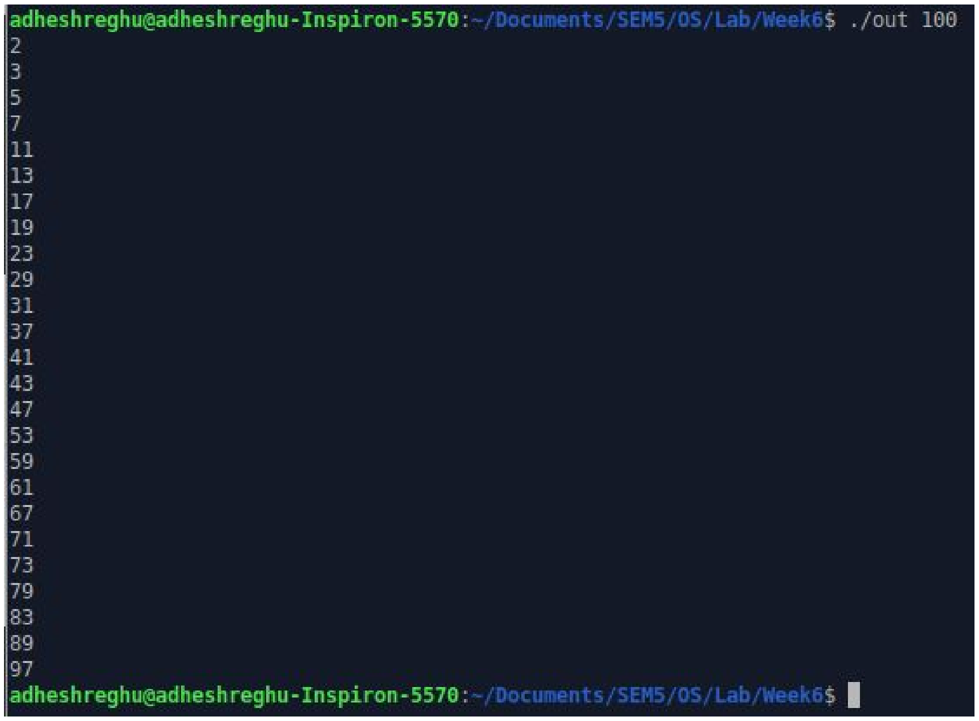
```c
}
// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
       // In case of '-'' symbol, continue if not the last character in string.
       if(character == 45 && i<strlen(str)-1)
             continue;
       // Else get the decimal value (-48) and check condition.
       character -= 48;
       if(character < 0 || character > 9)
             return 0;
    }
    return 1;
}

void *runner(void *arg)
{
    // Cast arg to integer
    int n = *((int *) arg);

    // Declare the return value
    int *ret = malloc(sizeof(int));
    *ret = 0;

    // Check primality of n
    *ret = isPrime(n);

    // Exit the thread and return primality status
    pthread_exit(ret);
}

// isPrime Utility Function
int isPrime(int n)
{
      // Corner cases
      if (n <= 1)
      return 0;
      if (n <= 3)
       return 1;
```

```
	if (n%2 == 0 || n%3 == 0)
	 return 0;

	int i;
	for (i=5; i*i<=n; i=i+6)
	if (n%i == 0 || n%(i+2) == 0)
		return 0;

	return 1;
}
```

Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ gcc -o out 4_prime.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 13
2
3
5
7
11
13
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 100
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```

5. Implement using threads : Computation of Mean, Median and Mode for an array of integers.

Logic Used

In this program, we wish to compute the mean, median and mode for a given array of integers each in a multi-threaded fashion. First let us take a look at what each of the terms mean.

Mean

Mean = (Sum of the integers in the array)/(Length of array)

This can be computed in a multi-threaded fashion by dividing the entire array into a certain number of partitions (decided by a global variable num_splits) and then computing the sum of each partition parallely in different threads. Each thread updates the sum to a global sum variable. Once all the threads finish computing the sum, we divide sum_arr (the global sum variable) with the length of the array to get the mean of the array.

Median

Median is defined as the middle element of a sorted list of numbers. So to compute the median we should sort the array. The sorting can be done parallely using a multi-threaded version of merge sort.

Mode

Mode is defined as the integer with the highest frequency in the given array. Since there is no upper bound defined on the array, computing the mode with hash wouldn't be wise. So instead, we can sort the array (parallely) and find the mode using a linear time scan of the array.

Since to find median and mode we are required to sort the array, the sorting can be done parallely along with the computation of the mean. So we compute the mean in one runner thread mean_runner whereas we sort the array using another thread mergeSort_thread. Once the array is sorted we join both the threads to the main program and feed the sorted array to two other new threads which computes the median and mode parallely - median_runner and mode_runner.

In this way, we can compute the mean, median and mode in a parallel fashion.

Let us take a look at the code.

Code

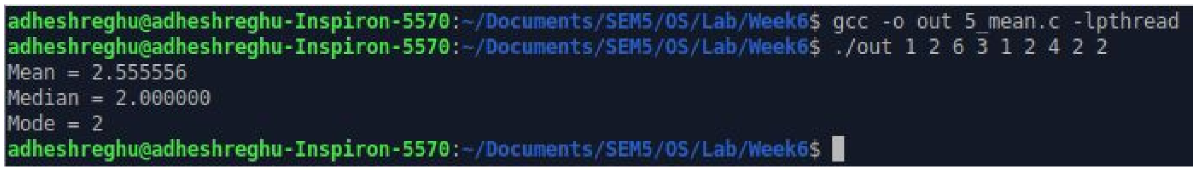
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <string.h>
#define MAX 100

// Define global parameters
long int sum_arr = 0;
int num_splits = 4, arr_len;

// allows the array to be passed by value (not by reference)
struct arrayWrapper
{
    int arr[MAX];
    int len;
    int split_no;
};

void *mean_runner(void *arg);
void *meanUtils_runner(void *arg);

// global declaration of arrayWrapper
struct arrayWrapper *A;

// struct to be used for merge sort
struct Position
{
    int start;
    int end;
};

void *mergeSort_thread(void *arg);
void merge(int begin, int mid, int end);
void *median_runner(void *arg);
void *mode_runner(void *arg);
```

```c
int main(int argc, char *argv[])
{
    A = (struct arrayWrapper *)malloc(sizeof(struct arrayWrapper));

    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc - 1;
       int i=1,n = 0;
       for(;i<=len;++i)
       {
       // Validation function. Warning if not integer.
       if(isInteger(argv[i]))
             A->arr[n++] = atoi(argv[i]);
       else
             printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i]);
       }

       // Update lengths
       A->len = n;
       arr_len = n;

       // Warning if entire argument list is invalid.s
       if(n == 0 )
             printf("The input list is invalid. Please enter an integer
array.\n");
       else
       {
             // Create two threads - one for computing mean and other for sorting
             pthread_t tid[2];
             // Call the mean thread
             pthread_create(&tid[0],NULL,mean_runner,NULL);

             // Define the struct for the start and end positions to be passed
             struct Position *pos = (struct Position *)malloc(sizeof(struct
Position));
             pos->start = 0;
             pos->end = n-1;
             // Call the sort thread
             pthread_create(&tid[1],NULL,mergeSort_thread,pos);
```

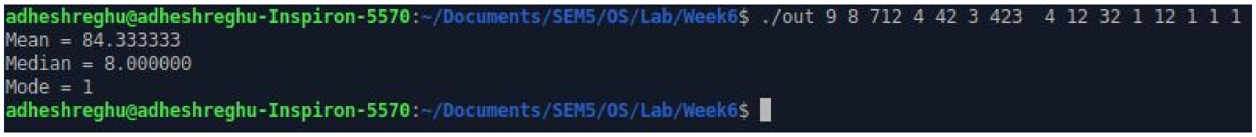
```c
            // Join the threads
            pthread_join(tid[0],NULL);
            pthread_join(tid[1],NULL);

            // Call the median and mode function in two different threads
            pthread_create(&tid[0],NULL,median_runner,NULL);
            pthread_create(&tid[1],NULL,mode_runner,NULL);

            // Join both the threads
            float *median;
            pthread_join(tid[0],(void*)&median);

            int *mode;
            pthread_join(tid[1],(void*)&mode);

            // Display mean, median and mode
            printf("Mean = %lf\n",(double)sum_arr/(double)arr_len);
            printf("Median = %lf\n",*median);
            printf("Mode = %d\n",*mode);
        }
    }
    return 0;
}

// Main runner function to compute the Mean
void *mean_runner(void *arg)
{
    /* -- Sum the entire array in threaded fashion by simming each split in a
different thread -- */
    if(arr_len < num_splits)
       num_splits = 2;

    // Declare the threads
    pthread_t tid[num_splits];

    int split_no = 0;
    int split_size = arr_len/num_splits;

    int i;
    struct arrayWrapper B[num_splits];
    for(i=0;i<num_splits;++i)
    {
        B[i] = *A;
        B[i].split_no = split_no;
        pthread_create(&tid[i],NULL,meanUtils_runner,(void *)&B[i]);
        ++split_no;
    }

    for(i=0;i<num_splits;++i)
        pthread_join(tid[i],NULL);

    pthread_exit(NULL);
}

// sub thread runner to compute mean
void *meanUtils_runner(void *arg)
{
    // Find the start and end of array split
    struct arrayWrapper *A = (struct arrayWrapper*)arg;
    int start = (arr_len * A->split_no)/num_splits;
    int end = (arr_len * (A->split_no + 1))/num_splits;

    // Define a local sum
    long int local_sum = 0;

    // Loop to find sum
    int i;
    for(i=start;i<end;++i)
        local_sum = local_sum + A->arr[i];

    // Update global sum
    sum_arr = sum_arr + local_sum;

    // Exit the thread
    pthread_exit(NULL);
}

// Define the runner function of the thread - mergeSort_thread
void *mergeSort_thread(void *arg)
{
    int begin = ((struct Position*)arg)->start;
    int end = ((struct Position*)arg)->end;

    if(begin < end)
    {
       int mid = begin + (end - begin)/2;

       // Create two threads - one for the LH partition and other for RH patition
       pthread_t tid[2];

       // Define the position struct for the Left Hand Half
       struct Position *leftPos = (struct Position *) malloc(sizeof(struct
Position));
       leftPos->start = begin;
       leftPos->end = mid;

       // Define the poisiton struct for the Right Hand Half
       struct Position *rightPos = (struct Position *)malloc(sizeof(struct
Position));
       rightPos->start = mid+1;
       rightPos->end = end;

       // Call the threads for both the halves
       pthread_create(&tid[0],NULL,mergeSort_thread,leftPos);
       pthread_create(&tid[1],NULL,mergeSort_thread,rightPos);

       // Join both the threads
       pthread_join(tid[0],NULL);
       pthread_join(tid[1],NULL);

       // call the merge function
       merge(begin,mid,end);
    }

    // Exit the thread
    pthread_exit(NULL);
}

// Define the runner function for median function
void *median_runner(void *args)
{
    // Define a return pointer
    float *ret = malloc(sizeof(float));
    *ret = 0.0;

    if(arr_len % 2 == 1)
       *ret = A->arr[arr_len/2];
    else
```

```
        *ret = (A->arr[(arr_len-1)/2] + A->arr[arr_len/2])/2.0;

    pthread_exit(ret);
}

// Define the runnner function for the mode function
void *mode_runner(void *args)
{
    // Define the return pointer
    int *ret = malloc(sizeof(int));
    *ret = 0;

    int i=0, max_cnt = 0, curr = A->arr[0], cnt = 0;
    for(i=1;i<arr_len;++i)
    {
        if(A->arr[i]!= curr)
        {
            if(cnt > max_cnt)
            {
                max_cnt = cnt;
                *ret = curr;
            }
            curr = A->arr[i];
            cnt = 0;
        }
        else if(A->arr[i] == curr)
            ++cnt;
    }
    if(max_cnt == 0)
        *ret = curr;
    pthread_exit(ret);
}

// Define the merge function
void merge(int begin, int mid, int end)
{
    int lsize = mid - begin + 1;
    int rsize = end - mid;

    // create left and right arrays
    int left[lsize], right[rsize];

    // fill in the arrays
    int i=0,j=0,k=0;
    for(i=0;i<lsize;++i)
       left[i] = A->arr[begin + i];
    for(j=0;j<rsize;++j)
       right[j] = A->arr[mid+1+j];

    k = begin;
    i=0,j=0;

    while(i<lsize && j<rsize)
    {
       if(left[i] < right[j])
             A->arr[k] = left[i++];
       else
             A->arr[k] = right[j++];

       ++k;
    }

    // Get the remaining elements from both the arrays into the merged array
    while(i<lsize)
       A->arr[k++] = left[i++];

    while(j<rsize)
       A->arr[k++] = right[j++];
}
// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
       // In case of '-'' symbol, continue if not last character in string.
       if(character == 45 && i<strlen(str)-1)
             continue;
       // Else get the decimal value (-48) and check condition.
       character -= 48;
       if(character < 0 || character > 9)
             return 0;
    }
    return 1;
}
```

Output

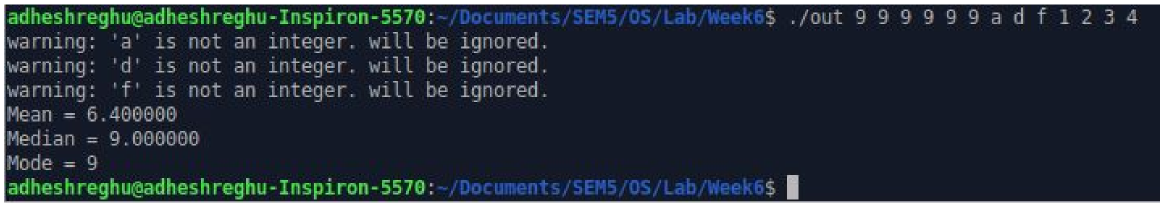


```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ gcc -o out 5_mean.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 1 2 6 3 1 2 4 2 2
Mean = 2.555556
Median = 2.000000
Mode = 2
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 9 8 712 4 42 3 423  4 12 32 1 12 1 1 1
Mean = 84.333333
Median = 8.000000
Mode = 1
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```

With invalid inputs



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 9 9 9 9 9 9 a d f 1 2 3 4
warning: 'a' is not an integer. will be ignored.
warning: 'd' is not an integer. will be ignored.
warning: 'f' is not an integer. will be ignored.
Mean = 6.400000
Median = 9.000000
Mode = 9
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```

6. Implement using threads : Implement merge sort and quick sort in a multi-threaded fashion.

Logic Used

In this program, we wish to implement merge sort and quick sort in a multi-threaded fashion.
First let's take a look at the merge sort implementation.

Merge Sort:
In the implementation of merge sort, we split the array to two partitions and sort each partition separately and then merge them after that. For a multi-threaded implementation, we can run the sort on each thread parallely in a different thread. We then join both the threads after sorting and merge the partitions. Since the mergeSort function is recursive the threads also get recursively created.
To implement the function, I have used a structure Position which maintains the start and end of each partition passed to the thread.

Let us take a look at the code

Code

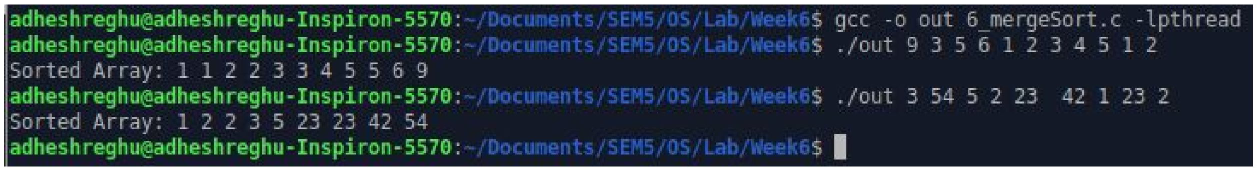
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <pthread.h>
#define MAX 1000

int isInteger(char* string);
void merge(int begin, int mid, int end);

// Define the array and length as global variable
int arr[MAX];
int n;

// Define a struct to pass the positional arguments to the thread
struct Position
{
    int start;
    int end;
};

void *mergeSort_thread(void *arg);

int main(int argc, char *argv[])
{

    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc - 1;
       int i=1;
       for(;i<=len;++i)
       {
       // Validation function. Warning if not integer.
       if(isInteger(argv[i]))
             arr[n++] = atoi(argv[i]);
       else
```

```
            printf("warning: '%s' is not an integer. will be ignored.\n",argv[i]);
        }

        // Warning if entire argument list is invalid.s
        if(n == 0 )
            printf("The input list is invalid. Please enter an integer array.\n");
        else
        {
            // Declare the main thread
            pthread_t tid;

            // Define the struct for the start and end positions to be passed
            struct Position *pos = (struct Position *)malloc(sizeof(struct
Position));
            pos->start = 0;
            pos->end = n-1;

            // Create the main thread
            pthread_create(&tid,NULL,mergeSort_thread,pos);

            // Join the array
            pthread_join(tid,NULL);

            // Print the array
            int i=0;
            printf("Sorted Array: ");
            for(;i<n;++i)
                printf("%d ",arr[i]);
            puts("");
        }
    }
    return 0;
}

// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
        character = str[i];
        // In case of '-'' symbol, continue if not last character in string.
        if(character == 45 && i<strlen(str)-1)
```

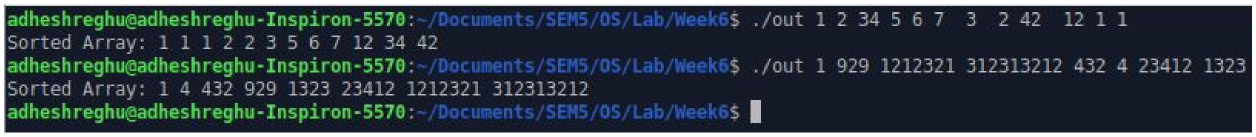
```
            continue;
        // Else get the decimal value (-48) and check condition.
        character -= 48;
        if(character < 0 || character > 9)
            return 0;
    }
    return 1;
}

// Define the runner function of the thread - mergeSort_thread
void *mergeSort_thread(void *arg)
{
    int begin = ((struct Position*)arg)->start;
    int end = ((struct Position*)arg)->end;

    if(begin < end)
    {
        int mid = begin + (end - begin)/2;

        // Create two threads - one for the LH partition and other for RH patition
        pthread_t tid[2];

        // Define the position struct for the Left Hand Half
        struct Position *leftPos = (struct Position *) malloc(sizeof(struct
Position));
        leftPos->start = begin;
        leftPos->end = mid;

        // Define the position struct for the Right Hand Half
        struct Position *rightPos = (struct Position *)malloc(sizeof(struct
Position));
        rightPos->start = mid+1;
        rightPos->end = end;

        // Call the threads for both the halves
        pthread_create(&tid[0],NULL,mergeSort_thread,leftPos);
        pthread_create(&tid[1],NULL,mergeSort_thread,rightPos);

        // Join both the threads
        pthread_join(tid[0],NULL);
        pthread_join(tid[1],NULL);

        // call the merge function
```

```c
        merge(begin,mid,end);
    }

    // Exit the thread
    pthread_exit(NULL);
}

// Define the merge function
void merge(int begin, int mid, int end)
{
    int lsize = mid - begin + 1;
    int rsize = end - mid;

    // create left and right arrays
    int left[lsize], right[rsize];

    // fill in the arrays
    int i=0,j=0,k=0;
    for(i=0;i<lsize;++i)
       left[i] = arr[begin + i];
    for(j=0;j<rsize;++j)
       right[j] = arr[mid+1+j];

    k = begin;
    i=0,j=0;

    while(i<lsize && j<rsize)
    {
       if(left[i] < right[j])
             arr[k] = left[i++];
       else
             arr[k] = right[j++];

       ++k;
    }

    // Get the remaining elements from both the arrays into the merged array
    while(i<lsize)
       arr[k++] = left[i++];

    while(j<rsize)
       arr[k++] = right[j++];
}
```

Output

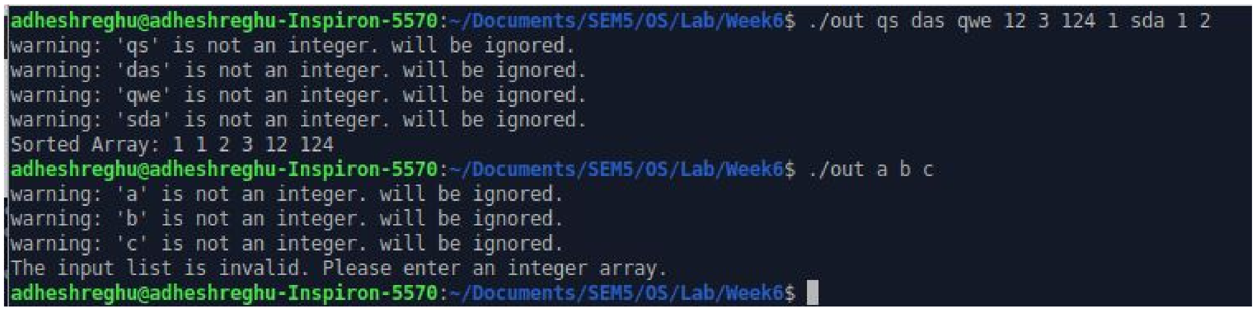


```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ gcc -o out 6_mergeSort.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 9 3 5 6 1 2 3 4 5 1 2
Sorted Array: 1 1 2 2 3 3 4 5 5 6 9
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 3 54 5 2 23  42 1 23 2
Sorted Array: 1 2 2 3 5 23 23 42 54
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 1 2 34 5 6 7  3  2 42  12 1 1
Sorted Array: 1 1 1 2 2 3 5 6 7 12 34 42
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 1 929 1212321 312313212 432 4 23412 1323
Sorted Array: 1 4 432 929 1323 23412 1212321 312313212
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```

With invalid inputs:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out qs das qwe 12 3 124 1 sda 1 2
warning: 'qs' is not an integer. will be ignored.
warning: 'das' is not an integer. will be ignored.
warning: 'qwe' is not an integer. will be ignored.
warning: 'sda' is not an integer. will be ignored.
Sorted Array: 1 1 2 3 12 124
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out a b c
warning: 'a' is not an integer. will be ignored.
warning: 'b' is not an integer. will be ignored.
warning: 'c' is not an integer. will be ignored.
The input list is invalid. Please enter an integer array.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```

Quick Sort:

Logic Used

In the implementation of quick sort, we first find the pivot, then sort each partition on the left and right of the pivot separately. For a multi-threaded implementation of the quick sort, we can first find the pivot position, then sort each partition on the left and right of the pivot in two different threads. Since quickSort function is a recursive function, the threads also are created recursively.
Here also we maintain a structure Position which stores the start and end position of each partition.

Let us take a look at the code

Code

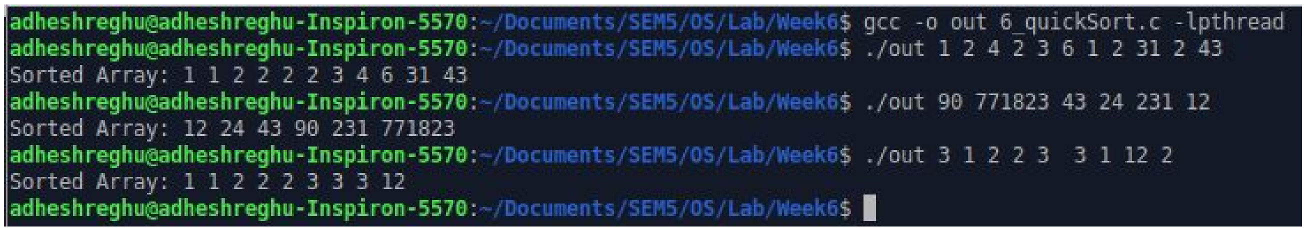
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <pthread.h>
#define MAX 1000

int isInteger(char* string);
int partition(int begin, int end);

// Define the array and length as global variable
int arr[MAX];
int n;

// Define a struct to pass the positional arguments to the thread
struct Position
{
    int start;
    int end;
};

void *quickSort_thread(void *arg);

int main(int argc, char *argv[])
{

    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc - 1;
       int i=1;
       for(;i<=len;++i)
       {
       // Validation function. Warning if not integer.
       if(isInteger(argv[i]))
             arr[n++] = atoi(argv[i]);
       else
```

```
            printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i]);
       }

       // Warning if entire argument list is invalid.s
       if(n == 0 )
             printf("The input list is invalid. Please enter an integer
array.\n");
       else
       {
             // Declare the main thread
             pthread_t tid;

             // Define the struct for the start and end positions to be passed
             struct Position *pos = (struct Position *)malloc(sizeof(struct
Position));
             pos->start = 0;
             pos->end = n-1;

             // Create the main thread
             pthread_create(&tid,NULL,quickSort_thread,pos);

             // Join the array
             pthread_join(tid,NULL);

             // Print the array
             int i=0;
             printf("Sorted Array: ");
             for(;i<n;++i)
                   printf("%d ",arr[i]);
             puts("");
       }
    }
    return 0;
}

// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
```

```
        // In case of '-'' symbol, continue if not last character in string.
        if(character == 45 && i<strlen(str)-1)
              continue;
        // Else get the decimal value (-48) and check condition.
        character -= 48;
        if(character < 0 || character > 9)
              return 0;
     }
     return 1;
}

// Define the runner function of the thread - mergeSort_thread
void *quickSort_thread(void *arg)
{
     int begin = ((struct Position*)arg)->start;
     int end = ((struct Position*)arg)->end;

     if(begin < end)
     {
        int p = partition(begin,end);

        // Create two threads - one for the LH partition and other for RH
patition
        pthread_t tid[2];

        // Define the position struct for the Left Hand Half
        struct Position *leftPos = (struct Position *) malloc(sizeof(struct
Position));
        leftPos->start = begin;
        leftPos->end = p-1;

        // Define the poisiton struct for the Right Hand Half
        struct Position *rightPos = (struct Position *)malloc(sizeof(struct
Position));
        rightPos->start = p+1;
        rightPos->end = end;

        // Call the threads for both the halves
        pthread_create(&tid[0],NULL,quickSort_thread,leftPos);
        pthread_create(&tid[1],NULL,quickSort_thread,rightPos);

        // Join both the threads
        pthread_join(tid[0],NULL);
```

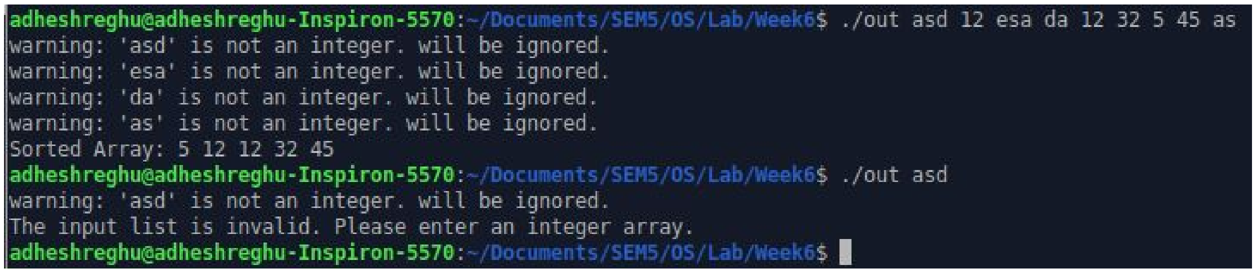
```c
       pthread_join(tid[1],NULL);
    }

    // Exit the thread
    pthread_exit(NULL);
}

// QuickSort - Partition Function
int partition(int begin, int end)
{
    int pivot = arr[end];
    int i = begin - 1,j,temp;
    for(j= begin;j<=end-1;++j)
    {
       if(arr[j] < pivot)
       {
             i++;
             // swap
             temp = arr[i];
             arr[i] = arr[j];
             arr[j] = temp;
       }
    }

    temp = arr[i+1];
    arr[i+1] = arr[end];
    arr[end] = temp;

    return i + 1;
}
```

Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ gcc -o out 6_quickSort.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 1 2 4 2 3 6 1 2 31 2 43
Sorted Array: 1 1 2 2 2 2 3 4 6 31 43
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 90 771823 43 24 231 12
Sorted Array: 12 24 43 90 231 771823
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out 3 1 2 2 3  3 1 12 2
Sorted Array: 1 1 2 2 2 3 3 3 12
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ 
```

With invalid inputs


```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out asd 12 esa da 12 32 5 45 as
warning: 'asd' is not an integer. will be ignored.
warning: 'esa' is not an integer. will be ignored.
warning: 'da' is not an integer. will be ignored.
warning: 'as' is not an integer. will be ignored.
Sorted Array: 5 12 12 32 45
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out asd
warning: 'asd' is not an integer. will be ignored.
The input list is invalid. Please enter an integer array.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```

7. Implement using threads : Estimation of PI Value using Monte Carlo simulation technique.

Logic Used

The idea is to simulate random (x, y) points in a 2-D plane with domain as a square of side 1 unit. Imagine a circle inside the same domain with the same diameter and inscribed into the square. We then calculate the ratio of number points that lie inside the circle and total number of generated points.
Since there is no relation between the random points generated, we can generate the random numbers in different threads. The thread runner can generate the random (x,y) and update the number of circle and square points (both global variables) based on the condition (if within the circle - the number of circle points gets updated).

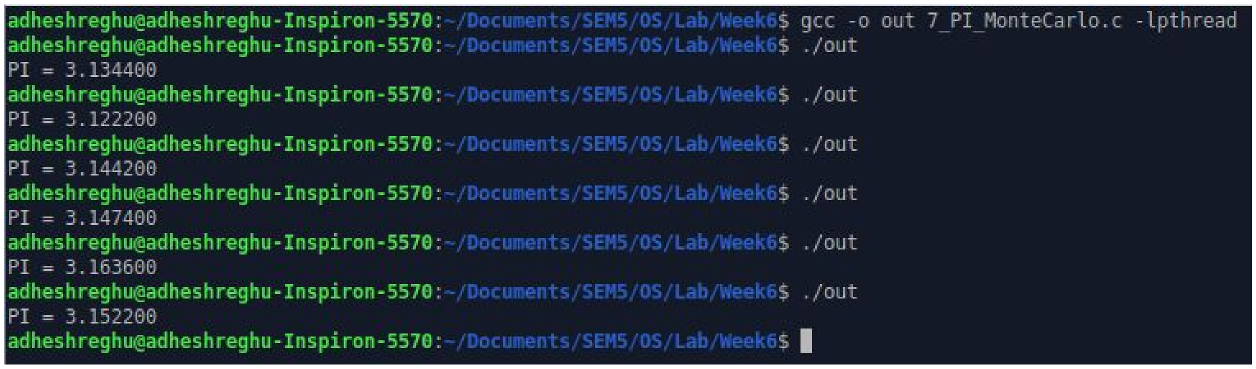
After generating INTERVAL number of random points we then estimate the value of PI as $4 * num\ of\ circle\ points\ /\ number\ of\ square\ points$.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>

// Let us define program MACROS
#define INTERVAL 20000

// define global variables
```

```c
long int num_circle_points = 0, num_sqr_points = 0;
double PI;

void *runner(void *arg);

int main()
{
    // Initialize the random number generator
    srand(time(NULL));

    // Define threads
    pthread_t tid[INTERVAL];

    // Loop
    int i = 0;
    for(;i<(INTERVAL);++i)
       pthread_create(&tid[i],NULL,runner,NULL);

    // Join all threads back
    i = 0;
    for(;i<(INTERVAL);++i)
       pthread_join(tid[i],NULL);

    // Print the value of PI
    PI  = (double)(4 * num_circle_points) / (double)num_sqr_points;
    printf("PI = %lf\n",PI);

    return 0;
}

void *runner(void *arg)
{
    double random_x, random_y;
    random_x = (double)(rand() % (INTERVAL + 1)) / (double)INTERVAL;
    random_y = (double)(rand() % (INTERVAL + 1)) / (double)INTERVAL;

    double d = random_x * random_x + random_y * random_y;

    if(d <= 1)
       ++num_circle_points;

    ++num_sqr_points;
    // Exit thread
    pthread_exit(NULL);
}
```

Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ gcc -o out 7_PI_MonteCarlo.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
PI = 3.134400
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
PI = 3.122200
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
PI = 3.144200
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
PI = 3.147400
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
PI = 3.163600
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
PI = 3.152200
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ 
```

8. Implement using threads : Computation of inverse of a matrix using determinants and cofactors.

Logic Used

In this question, we are required to compute the inverse of the matrix using determinants and adjoint (using cofactors).
To compute the inverse we need to compute the adjoint and the determinant of the input matrix and then

Inverse A = $1/det * Adjoint$ (A)

So, to achieve parallelism using threads we can compute the determinant and adjoint parallely in two different threads.
The determinant runner thread - det_runner is a recursive function and hence threads also get created recursively.
The adjoint runner thread - adj_runner also uses determinant function, so it also calls for the det_runner by creating threads within it.

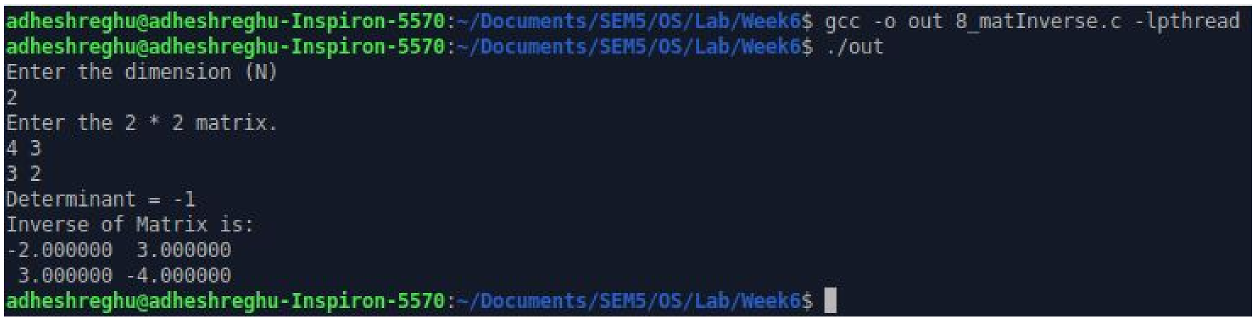
Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>

#define MAX 100

// declare an array structure
struct arrayWrapper
{
    double arr[MAX][MAX];
    int N;
    int n;
};

// Declare global variables
struct arrayWrapper *A;
struct arrayWrapper *adj;
int N;

void *det_runner(void *arg);
void *adjoint_runner(void *arg);

int main()
{
    A = (struct arrayWrapper *)malloc(sizeof(struct arrayWrapper));
    adj = (struct arrayWrapper *)malloc(sizeof(struct arrayWrapper));

    // Accept input
    printf("Enter the dimension (N)\n");
    scanf("%d",&N);
    A->N = N;
    A->n = N;

    // Accept the matrix
    printf("Enter the %d * %d matrix.\n",A->N,A->N);
    int i,j;
    for(i=0;i<N;++i)
    {
       for(j=0;j<N;++j)
       {
             scanf("%lf",&(A->arr[i][j]));
       }
    }

    // Create two threads to compute the adjoint and determinant parallelely
    pthread_t tid[2];

    // Call the thread - to compute determinant
    struct arrayWrapper B  = *A;
    pthread_create(&tid[0],NULL,det_runner,(void *)&B);

    // Call the thread to compute the adjoint
    adj->N = N;
    pthread_create(&tid[1],NULL,adjoint_runner,NULL);

    // Join both the threads
    int *det;
    pthread_join(tid[0],(void*)&det);
    printf("Determinant = %d\n",*det);

    // Join adj thread
    pthread_join(tid[1],NULL);

    // Print the inverse matrix
    printf("Inverse of Matrix is:\n");
    for(i=0;i<N;++i)
    {
       for(j=0;j<N;++j)
             printf("%9lf ",adj->arr[i][j]/(float)*det);
       puts("");
    }

    return 0;
}

// Define the runner thread to compute the determinant
void *det_runner(void *arg)
{
    // Cast the arg
    struct arrayWrapper *B = (struct arrayWrapper*)arg;
    int n = B->n;

    // Define the return value
    int *det = malloc(sizeof(int));
    *det = 0;

    // Base case
    if(n == 1)
    {
       *det = B->arr[0][0];
       pthread_exit(det);
    }

    // Define variables
    int sign = 1;
    int i,j,k,row,col;

    // Define temp struct to store the cofactors in each loop
    struct arrayWrapper temp[n];

    for(i=0;i<n;++i)
    {
       // Compute Cofactor in place - passing to a function causes prolem
       j=0,k=0;
       for(row=0;row<n;++row)
       {
             for(col=0;col<n;++col)
             {
                   if(row!=0 && col!=i)
                   {
                         temp[i].arr[j][k++] = B->arr[row][col];

                         if(k == n-1)
                         {
                               k = 0;
                               ++j;
                         }
                   }
             }
       }
       temp[i].n = n-1;

       // Create a new thread to calc the determinant of the cofactor
```

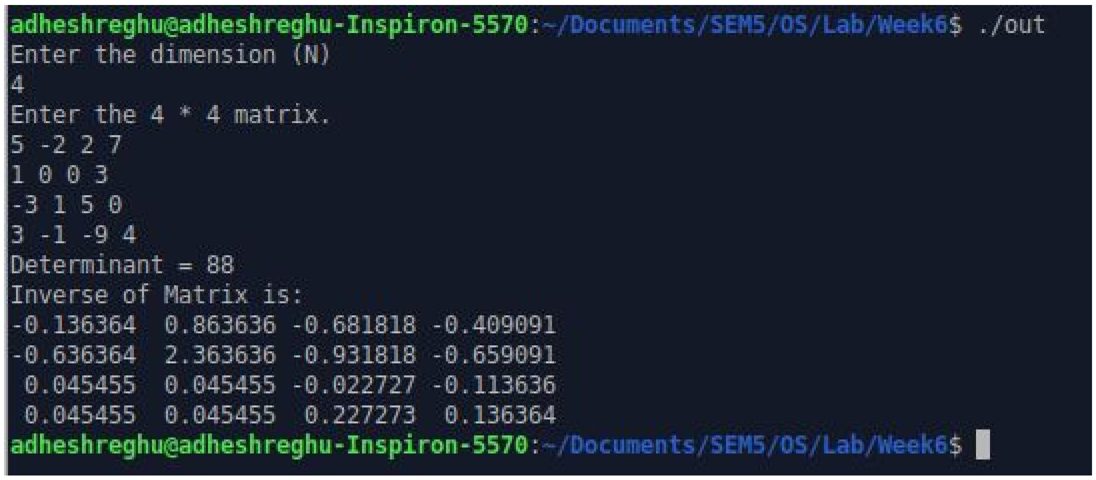
```
        pthread_t tid;
        pthread_create(&tid,NULL,det_runner,(void *)&temp[i]);

        // Join the thread and get return value
        int *local_det;
        pthread_join(tid,(void*)&local_det);

        // Compute the determinant
        *det = *det + sign * B->arr[0][i] * (*local_det);

        // Flip the sign
        sign = -1*sign;
    }

    // Exit and return the determinant.
    pthread_exit(det);
}

// Define the adjoint runner thread
void *adjoint_runner(void *arg)
{
    if(N == 1)
    {
        adj->arr[0][0] = 1;
        pthread_exit(NULL);
    }

    // Define cofactor array
    struct arrayWrapper cofactor[N][N];

    int sign = 1;

    // Loop
    int i,j,row,col,p,q;
    for(i=0;i<N;++i)
    {
        for(j=0;j<N;++j)
        {
                // Compute Co-factor inplace
                cofactor[i][j].N = N-1;
                cofactor[i][j].n = N-1;
                p=0,q=0;
                for(row=0;row<N;++row)
```

```
            {
                for(col=0;col<N;++col)
                {
                    if(row!=i && col!=j)
                    {
                        cofactor[i][j].arr[p][q++] = A->arr[row][col];

                        if(q == N-1)
                        {
                            q = 0;
                            ++p;
                        }
                    }
                }
            }

            // determine sign
            if((i+j)%2 == 0)
                sign = 1;
            else
                sign = -1;

            // Get the determinant of cofactor matrix by calling the determinant
thread
            int *det;
            pthread_t tid;
            pthread_create(&tid,NULL,det_runner,(void *)&cofactor[i][j]);

            pthread_join(tid,(void*)&det);

            adj->arr[j][i]  = sign * (*det);
        }
    }

    pthread_exit(NULL);
}
```

Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ gcc -o out 8_matInverse.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
Enter the dimension (N)
2
Enter the 2 * 2 matrix.
4 3
3 2
Determinant = -1
Inverse of Matrix is:
-2.000000  3.000000
 3.000000 -4.000000
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$ ./out
Enter the dimension (N)
4
Enter the 4 * 4 matrix.
5 -2 2 7
1 0 0 3
-3 1 5 0
3 -1 -9 4
Determinant = 88
Inverse of Matrix is:
-0.136364  0.863636 -0.681818 -0.409091
-0.636364  2.363636 -0.931818 -0.659091
 0.045455  0.045455 -0.022727 -0.113636
 0.045455  0.045455  0.227273  0.136364
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week6$
```

9. Implement using threads : Generation of nth Fibonacci number.

Logic Used

In this program we are required to generate the nth fibonacci number using multi-threading.
We cannot parallelize the bottom-up approach [O(n) method] as there are data dependencies. However we can parallelize the recursive version of the algorithm. Let us first take a look at the standard recursive algorithm.

```
long int fibonacci(int  n)
{
      if(n<=2)
      return 1;
      else
      return fibonacci(n-1) + fibonacci(n-2);
}
```

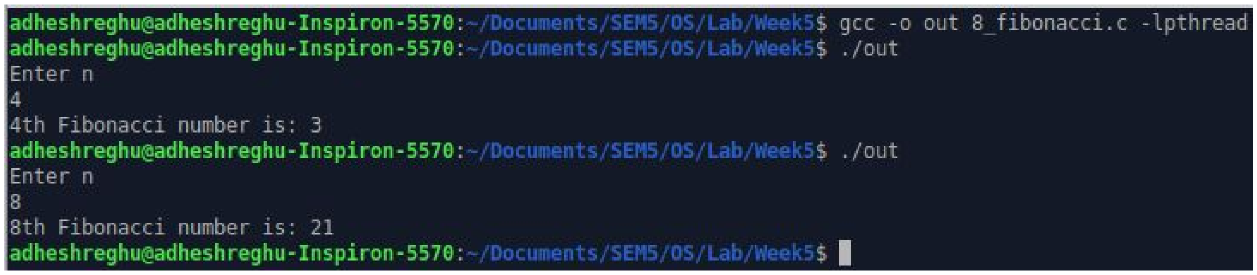
Here the fibonacci(n-1) and fibonacci(n-2) can be calculated in two different threads and their results can be added.
The runner function of the thread returns 1 when n<=2, otherwise it creates two more threads to calculate fibonacci(n-1) and fibonacci(n-2), it then waits for the result from the child threads and then combines the result to get the nth fibonacci number and returns it.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <inttypes.h>

void *fib_thread(void *arg);

int main()
{
    // Accept input n
    int n;
    printf("Enter n\n");
    scanf("%d",&n);

    // Generate nth fibonacci number
    int *arg = malloc(sizeof(*arg));
    *arg = n;

    pthread_t tid;
    pthread_create(&tid,NULL,fib_thread,arg);

    // Join back the thread
    int *fib_n;
    pthread_join(tid,(void*)&fib_n);

    printf("%dth Fibonacci number is: %d\n",n,*fib_n);

    return 0;
}
```

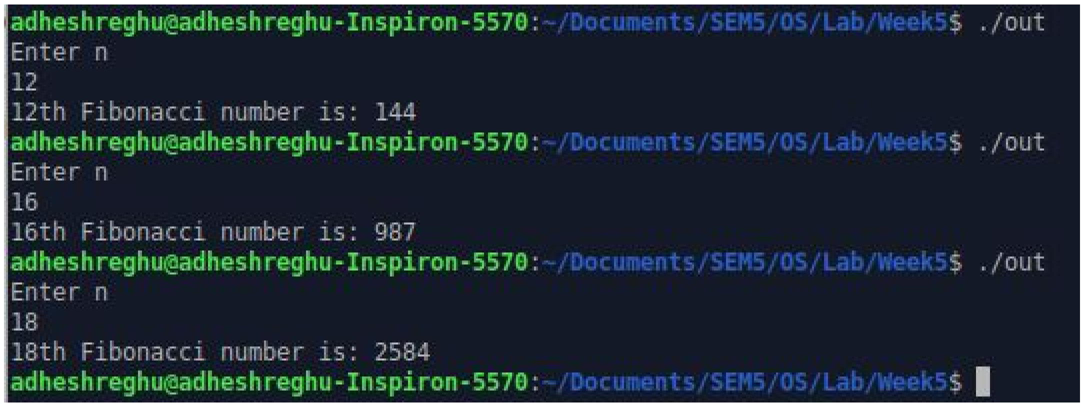
```c
void *fib_thread(void *arg)
{
    // cast to int
    int n = *((int *) arg);

    // Declare the return variable
    int *ret = malloc(sizeof(int));
    *ret = 0;

    if(n <= 2)
       *ret = 1;
    else
    {
       int *arg_1 = malloc(sizeof(*arg));
       int *arg_2 = malloc(sizeof(*arg));

       // Declare two pthreads
       pthread_t tid[2];

       // Declare two return integer pointers
       int *ret_1, *ret_2;

       // Create two threads for each recursion tree
       *arg_1 = n-1;
       pthread_create(&tid[0],NULL,fib_thread,arg_1);

       *arg_2 = n-2;
       pthread_create(&tid[1],NULL,fib_thread,arg_2);

       // Join both threads back
       pthread_join(tid[0],(void*)&ret_1);
       pthread_join(tid[1],(void*)&ret_2);

       // return the sum
       *ret = *ret_1 + *ret_2;
    }
    pthread_exit(ret);
}
```

Output



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 8_fibonacci.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
4
4th Fibonacci number is: 3
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
8
8th Fibonacci number is: 21
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

More Examples:



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
12
12th Fibonacci number is: 144
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
16
16th Fibonacci number is: 987
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
18
18th Fibonacci number is: 2584
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

10. Implement using threads : Find the length of the longest common subsequence of two strings.

Logic Used

In this program we are required to find the length of the longest common subsequence of two strings in a multi-threaded fashion. We first accept both the strings and store them in global character arrays s1 and s2.
First let us take a look at a naive recursive algorithm to find the length of the longest common subsequence of two strings.

```c
int lcs(int m, int n )
{
   if (m == 0 || n == 0)
     return 0;
   if (s1[m-1] == s2[n-1])
     return 1 + lcs(m-1, n-1);
```

```
  else
    return max(lcs(m, n-1), lcs(m-1, n));
}
```

Clearly we can parallelize the recursive algorithm by implementing the algorithm in the runner function of the thread - lcs_thread. In this runner function, each call of lcs would create a new thread with arguments m and n passed to the runner of the thread. Since we have to pass two arguments to each runner function of thread, we use a structure called Param. Let us take a look at the structure.

```
// structure to pass positional values for two strings
struct Param
{
    int m,n;
};
```

In the runner function we calculate the return value and pass it back to the calling function (either a parent thread or main) via pthread_exit(ret) call. Once we get the return value to the main (after exit of the first thread), we print the result and end the program.

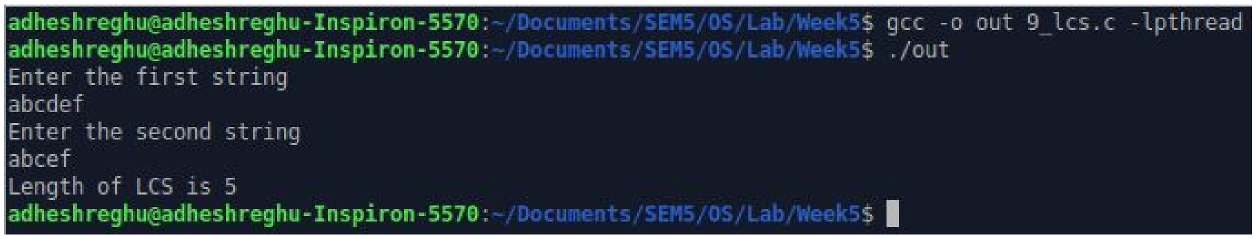
Code

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <string.h>
#define MAX 1000


// structure to pass positional values for two strings
struct Param
{
    int m,n;
};
```

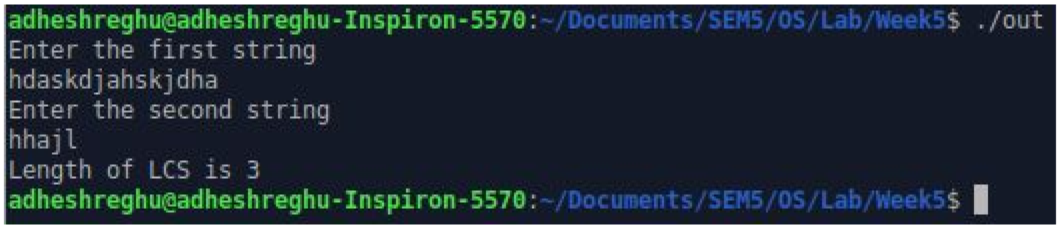
```c
// Global declaration of the two strings
char s1[MAX], s2[MAX];

void *lcs_thread(void *arg);
int max(int a, int b);

int main()
{
    // Accept both strings as input
    printf("Enter the first string\n");
    scanf("%s",s1);

    printf("Enter the second string\n");
    scanf("%s",s2);

    // Instantiate a struct
    struct Param *param = (struct Param*)malloc(sizeof(struct Param));
    param->m = strlen(s1);
    param->n = strlen(s2);

    // Create a main thread (initial thread)
    pthread_t tid;
    pthread_create(&tid,NULL,lcs_thread,param);

    // Define return value from thread
    int *max_len;
    pthread_join(tid,(void*)&max_len);

    printf("Length of LCS is %d\n",*max_len);

    return 0;
}

// Define the LCS Thread
void *lcs_thread(void *arg)
{
    int m = ((struct Param*)arg)->m;
    int n = ((struct Param*)arg)->n;

    // Define return variable
    int *ret = malloc(sizeof(int));
    *ret = 0;
    // Trivial condition
    if(m == 0 || n == 0)
       pthread_exit(ret);

    if(s1[m-1] == s2[n-1])
    {
       // Define a new struct copied from the arg
       struct Param arg_1 = *((struct Param*)arg);
       --arg_1.m;
       --arg_1.n;

       // Recursively call the thread again
       pthread_t tid;
       pthread_create(&tid,NULL,lcs_thread,(void *)&arg_1);

       // Get return value from thread
       int *ret_1;
       pthread_join(tid,(void*)&ret_1);
       *ret = *ret_1 + 1;
    }
    else
    {
       // Define new structs copied from the arg
       struct Param arg_1 = *((struct Param*)arg);
       --arg_1.n;
       struct Param arg_2 = *((struct Param*)arg);
       --arg_2.m;

       // Recursively call the thread again
       pthread_t tid[2];
       pthread_create(&tid[0],NULL,lcs_thread,(void *)&arg_1);
       pthread_create(&tid[1],NULL,lcs_thread,(void *)&arg_2);

       // Get return value from both threads
       int *ret_1, *ret_2;
       pthread_join(tid[0],(void*)&ret_1);
       pthread_join(tid[1],(void*)&ret_2);

       *ret = max(*ret_1,*ret_2);
    }
    pthread_exit(ret);
}
```

```c
int max(int a, int b)
{
    if(a > b)
       return a;
    return b;
}
```

Output

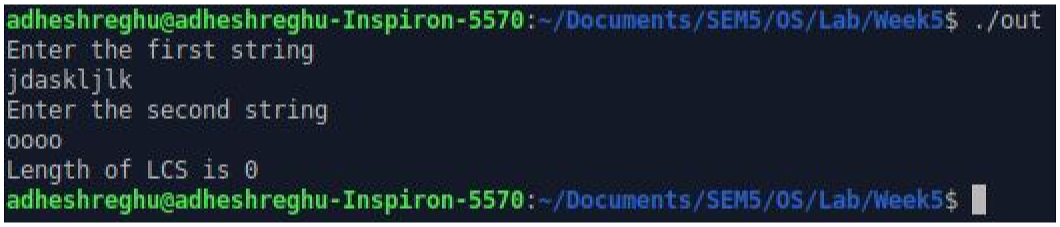


```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 9_lcs.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the first string
abcdef
Enter the second string
abcef
Length of LCS is 5
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the first string
hdaskdjahskjdha
Enter the second string
hhajl
Length of LCS is 3
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: When then length of lcs is zero.



```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the first string
jdaskljlk
Enter the second string
oooo
Length of LCS is 0
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```