# Assignment 2

Hello. Your next assignment involves 3 programming questions; 2 using MPI and 1 through RPC.
Kindly refer to the other uploaded document for your 3 questions.
You can refer to the links posted previously for any help on MPI and RPC or get in touch with us.
Only C++ is allowed for this assignment.

Upload all the codes and other files in a single folder as *rollnumber.zip*.
For MPI programs:

mpi_questionno.cpp (ex: mpi_12.cpp)

For RPC:

prog.x
client.cpp
server.cpp

For MPI programs, read from stdin and print to stdout.

## MPI Part A

### 1. IsPrime

Given a natural number N, output whether N is prime or not.
(N will fit into a 32-bit integer)

INPUT: 2
OUTPUT: Yes

INPUT: 100
OUTPUT: No

### 2. Vertex Coloring

Given an undirected graph G, assign a color to every vertex such that no two adjacent vertices have the same color.
Total number of colors should not exceed d + 1 where d is the maximum degree of the graph.

First line of input contains two integers n and m. This is followed by m lines where each line is for an edge between 2 nodes in G.

Output the number of colors used in the first line followed by n lines printing the color of nodes from 0 to n - 1.

```
INPUT:
6 7
0 1
0 2
1 3
2 3
3 4
3 5
4 5

OUTPUT:
3
0
1
1
0
1
2
```

Note: Your output may be different but ensure that it follows the mentioned constraints.

**3. Edge Coloring**

Given an undirected graph G, assign a color to each edge such that no two adjacent edges have the same color.
Total number of colors should not exceed d + 1 where d is the maximum degree of the graph.

First line of input contains two integers n and m. This is followed by m lines where each line is for an edge between 2 nodes in G.

Output the number of colors used in the first line followed by m lines printing the color of each edge as given in input.

```
INPUT:
6 7
0 1
0 2
1 3
2 3
3 4
```

```
3 5
4 5

OUTPUT:
4
1
0
0
1
2
3
0
```

Note: Your output may be different but ensure that it follows the mentioned constraints.

**4. Sorting**

Given N distinct numbers, output them in sorted order.

```
INPUT
7
3 5 1 2 4 7 6

OUTPUT
1 2 3 4 5 6 7
```

**5. k$^{th}$ Smallest Element**

Given N distinct numbers and k <= N, output the k$^{th}$ smallest number.

```
INPUT
7 2
3 5 1 2 4 7 6

OUTPUT
2
```

**6. sin(x)**

Given x and n, calculate taylor expansion of sin(x) till first n terms.
Output your answer rounded to 2 decimals.

**7. cos(x)**

Given x and n, calculate taylor expansion of cos(x) till first n terms.
Output your answer rounded to 2 decimals.

## MPI Part B

### 8. Matrix Inverse via Row Reduction

Given an square invertible matrix A, output its inverse using only elementary row operations.

```
INPUT
3
2 3 4
5 4 2
1 1 1
OUTPUT
-2 -1 -10
3 2 -16
-1 -1 7
```

Round numbers in inverse matrix to 2 decimals.

### 9. Matrix Determinant via Row Reduction

Given an square matrix A, output its determinant using only elementary row operations.

```
INPUT
3
1 2 6
8 9 11
5 4 7

OUTPUT
-61
```

You can assume the answer will fit in an integer.

### 10. Gaussian elimination.

Output the solution of a set of simultaneous linear equations using Gaussian elimination.

INPUT
2 (Number of variables)
5 2 10 ($5x_1 + 2x_2 = 10$)

1 3 7 ($x_1 + 3x_2 = 7$)

OUTPUT
1.23 ($x_1 = 16/13$)
1.92 ($x_2 = 25/13$)

Round off your answer to 2 decimals.

**11. Word Grouping**

Given a input text file, output the frequency of each word in order of increasing frequency.
If multiple words have the same frequency use default C++ string sort for sorting.
You can assume file to be a combination of words and spaces.

```
INPUT
The     the cake IS a lie cake Lie Lie lie x

OUTPUT
a 1
is 1
x 1
cake 2
the 2
lie 4
```

You can assume the file to have no special characters.

**12 Triangles in a Graph**

Given an undirected graph G, output the total number of triangles in G.
First line of input would be 2 integers, n and m followed by m lines of edges.

```
INPUT
5 7
0 1
0 2
1 2
1 3
2 3
1 4
2 4
```

```
OUTPUT
3
```

**13. 4-cycles in a Graph.**

Given an undirected graph G, output the total number of 4-vertex cycles in G.
First line of input would be 2 integers, n and m followed by m lines of edges.

```
INPUT
5 7
0 1
0 2
1 2
1 3
2 3
1 4
2 4

OUTPUT
3
```

**14. 2-Distance Coloring**

Given an undirected graph G, color the graph such that all vertices which are adjacent or share a common neighbor receive different colors.
You can assume that the graph can be colored by d*d colors where d is the maximum degree of the graph.
First line of input contains two integers n and m. This is followed by m lines where each line is for an edge between 2 nodes in G. Output n lines denoting the colors for n vertices in order.

```
INPUT
8 9
0 1
0 2
0 3
1 4
2 5
3 6
4 7
5 7
6 7

OUTPUT
0
```

```
1
2
3
2
3
1
0
```

Note: Different Outputs are allowed as long as they are correct.

## RPC

System calls are allowed. Output must match bash output.
Use man pages if you are unfamiliar with the native bash command.
Run these commands on server side!

**1. rwho**

**Format :** rwho <*arguments*>

Should display who is logged in on remote server.

**Arguments :** It should support following argument for rwho (Read about these arguments in man page of who command).

- ***-a***

**Example :** *rwho*

*rwho -a*

*If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of* ***rwho*** *unless the* ***-a*** *flag is given.*

**2. rgrep**

**Format :** rgrep <*arguments*> <*pattern*> <*file*>

Searches for pattern in each file on a remote server. By default, rgrep should display the matching lines.

**Arguments :** It should support following arguments (Read about these arguments in man page of grep command).

- ***-i***
- ***-c***

**Example :** *rgrep -i <pattern> <file>*

(Ignore case distinctions, so that characters that differ only in case match each other.)

*rgrep -c <pattern> <file>*

(Suppress normal output; instead print a count of matching lines for input file.)

**3. rls**

**Format :** *ls <path> <arguments>*

Should display the ls output for the given path in the remote server. If no path is given show the contents of the folder from which the server is started (CWD).

Arguments are optional.

**Arguments :** It should support following three arguments for ls (Read about these arguments in man page of ls command).

- ***-a***
- ***-l***
- ***-i***

**Example :** *rls*
(Contents of Current working directory inside server)
*rls -l*
(Contents of Current working directory inside server in list format)

**4. rlocate**

**Format :** *rlocate <pattern> <arguments>*
Should display the output of the *locate* command with given pattern and arguments. Arguments are optional.
**Arguments :** It should support following three arguments for *locate* (Read about these arguments in man page of *locate* command).

- ***-A***
- ***-b***
- ***-c***

**Example :** *rlocate hello*
(*locate* writes file names matching the *<pattern>* to standard output, one per line. This should be executed on the remote server and output should be displayed in the client)

**5. rman**

**Format :** *rman <arguments> <program/utility/function>*
Should display the system's reference manuals.

**Example :** *rman ls*
(Contents of ls man page as per bash should be displayed)

**6. rfinger**

**Format :** *rfinger <arguments>*
Should display information about computer users such as login name, the full name, and possibly other details about the user.

**Example :** *rfinger*
(Login     Name       Tty       Idle  Login Time   Office      Office Phone)

**7. rlast**:

Format: $./rlast <arguments>
Where <arguments> is the set of all the valid arguments for the linux command *last.*

The client prints the output of the command on stdout as it would on the server machine.

**8. rapropos**:

Format: $./rapropos <arguments>
Where <arguments> is the set of all the valid arguments for the linux command *apropos.*

The client prints the output of the command on stdout as it would on the server machine.

**9. rhelp:**

**Format:** $ ./rhelp valid_linux_command

Most linux command have a --help argument, which gives a quick reference to all the valid arguments for that command. You are to implement a remote version of this feature. The client prints the output of the command on stdout as it would on the server machine.

**Example:**

<Client>$ ./rhelp ls

should be *same as*

<Server>$ ls --help