Name: Sadaf Najeem Mulla

PRN No: 2019BTECS00038

**High Performance Computing Lab**

**Practical No. 5**

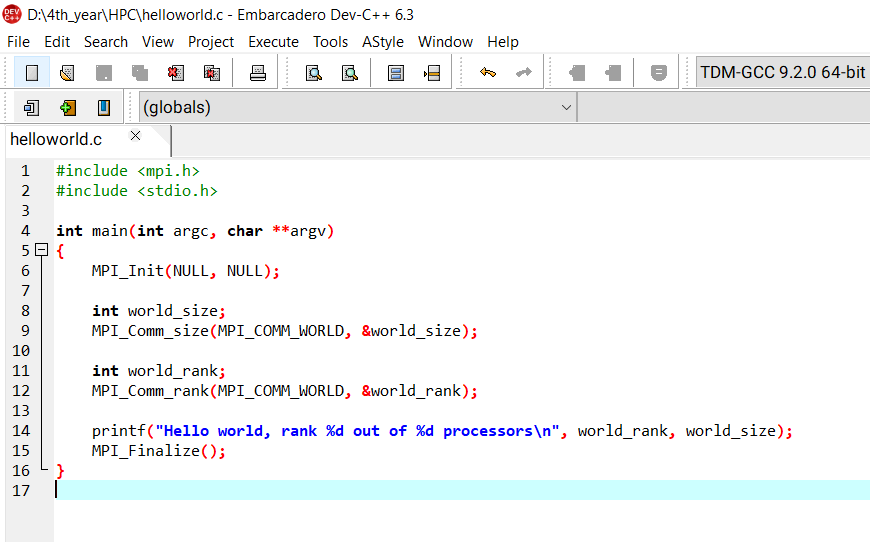
**Title of practical:**

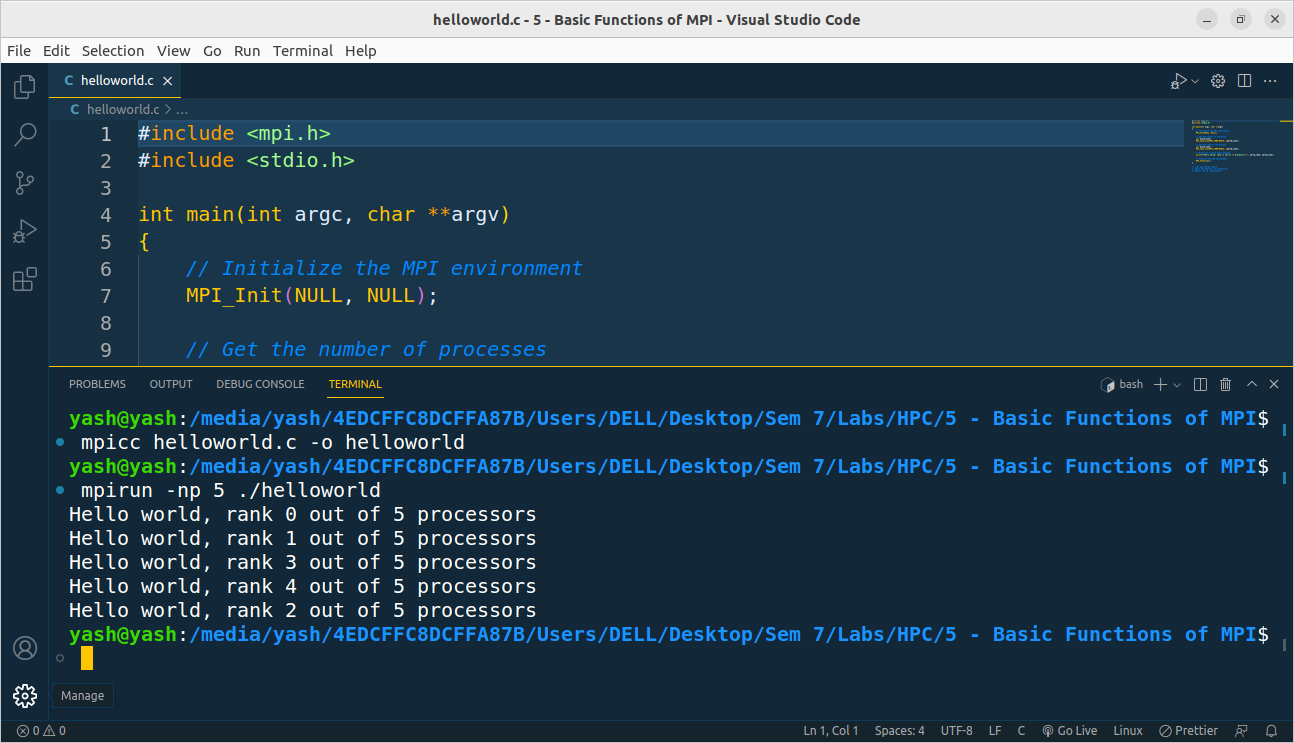
Installation of MPI & Implementation of basic functions of MPI

**Problem Statement 1:**

Implement a simple hello world program by setting number of processes equal to 10

**Screenshots:**



****

**Information 1:**

First step to building an MPI program is including the MPI header files with #include <mpi.h>. After this, the MPI environment must be initialized with MPI\_Init.

During MPI\_Init, all of MPI’s global and internal variables are constructed. For example, a communicator is formed around all of the processes that were spawned, and unique ranks are assigned to each process. Currently, MPI\_Init takes two arguments that are not necessary, and the extra parameters are simply left as extra space in case future implementations might need them.

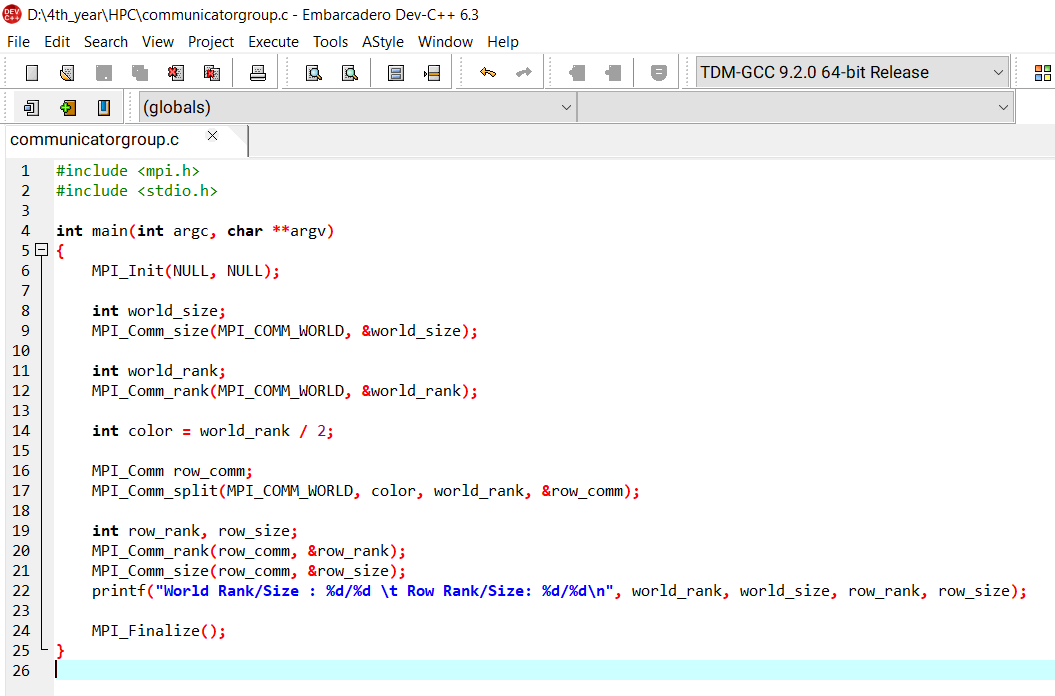
After MPI\_Init, there are two main functions that are called. These two functions are used in almost every single MPI program:

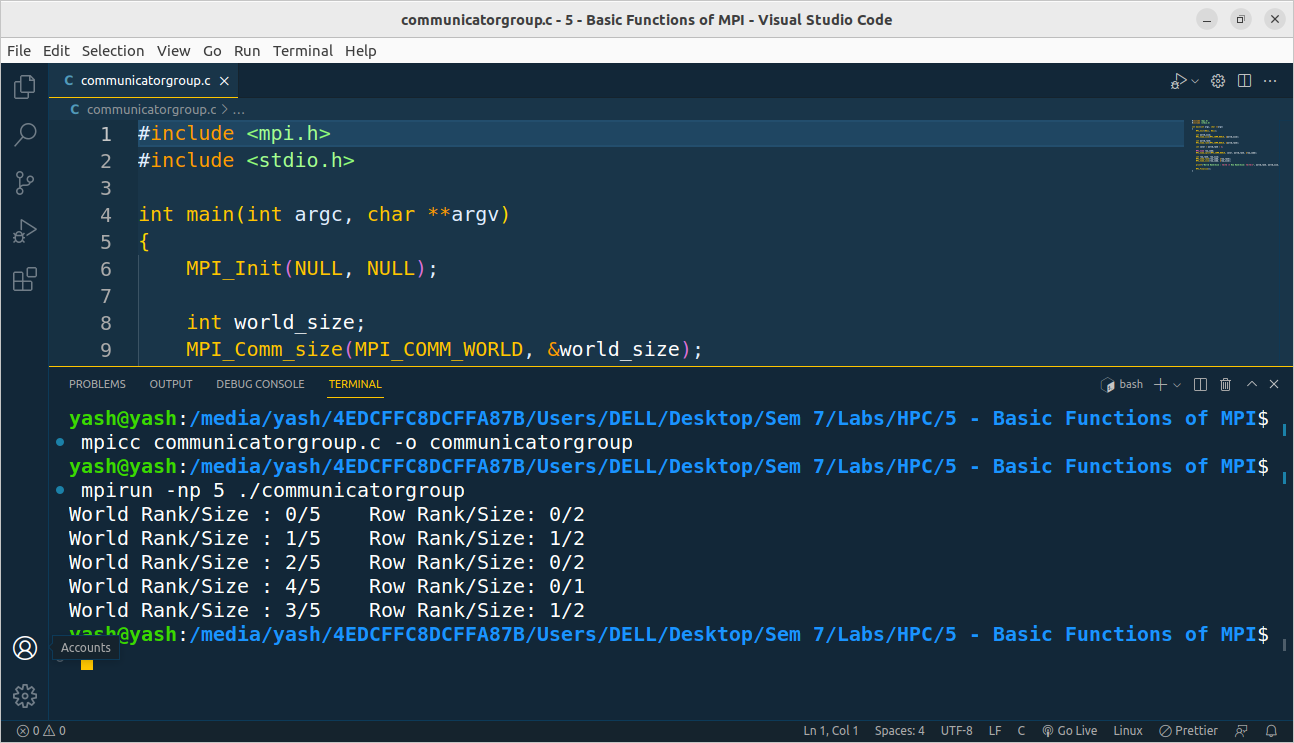
1. MPI\_Comm\_size returns the size of a communicator. In our example, MPI\_COMM\_WORLD (which is constructed for us by MPI) encloses all of the processes in the job, so this call should return the amount of processes that were requested for the job.
2. MPI\_Comm\_rank returns the rank of a process in a communicator. Each process inside of a communicator is assigned an incremental rank starting from zero. The ranks of the processes are primarily used for identification purposes when sending and receiving messages.

MPI\_Finalize is used to clean up the MPI environment. No more MPI calls can be made after this one.

**Problem Statement 2:**

Implement a program to display rank and communicator group of five processes



****

**Information 2:**

MPI\_Comm\_split creates new communicators by “splitting” a communicator into a group of sub-communicators based on the input values color and key. It’s important to note here that the original communicator doesn’t go away, but a new communicator is created on each process. The first argument, comm, is the communicator that will be used as the basis for the new communicators. This could be MPI\_COMM\_WORLD, but it could be any other communicator as well. The second argument, color, determines to which new communicator each processes will belong. All processes which pass in the same value for color are assigned to the same communicator. The third argument, key, determines the ordering (rank) within each new communicator. The process which passes in the smallest value for key will be rank 0, the next smallest will be rank 1, and so on. If there is a tie, the process that had the lower rank in the original communicator will be first. The final argument, newcomm is how MPI returns the new communicator back to the user.