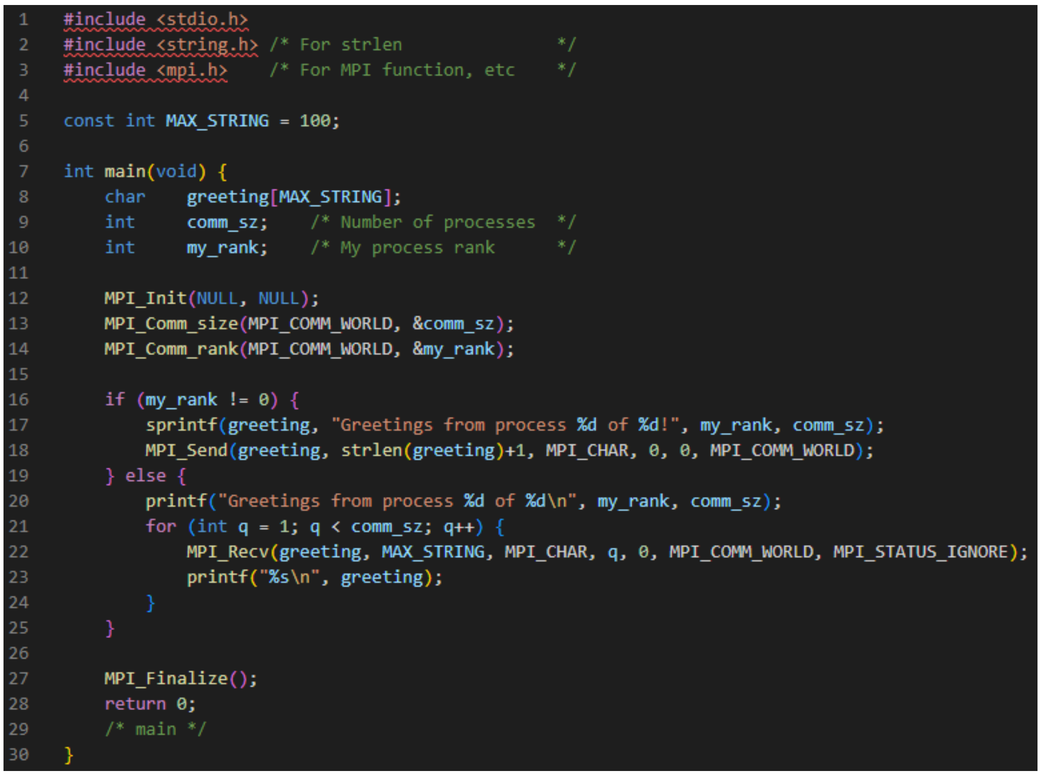
Assignment 1 Milestone

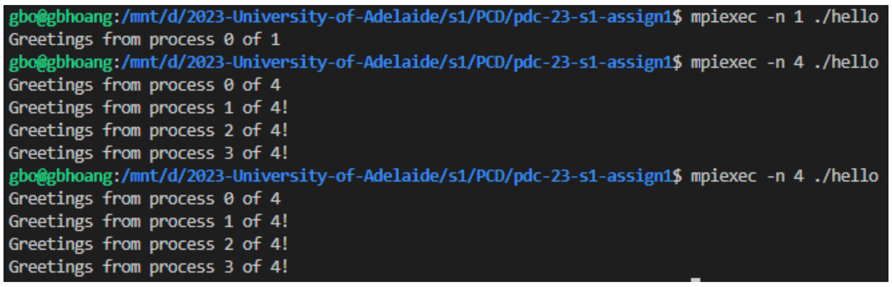
Gia Bao Hoang – a1814824

Step 1:



```c
#include <stdio.h>
#include <string.h> /* For strlen             */
#include <mpi.h>    /* For MPI function, etc    */

const int MAX_STRING = 100;

int main(void) {
    char     greeting[MAX_STRING];
    int      comm_sz;     /* Number of processes  */
    int      my_rank;     /* My process rank      */

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank != 0) {
        sprintf(greeting, "Greetings from process %d of %d!", my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    } else {
        printf("Greetings from process %d of %d\n", my_rank, comm_sz);
        for (int q = 1; q < comm_sz; q++) {
            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s\n", greeting);
        }
    }

    MPI_Finalize();
    return 0;
    /* main */
}
```

Figure 1: Program hello.c



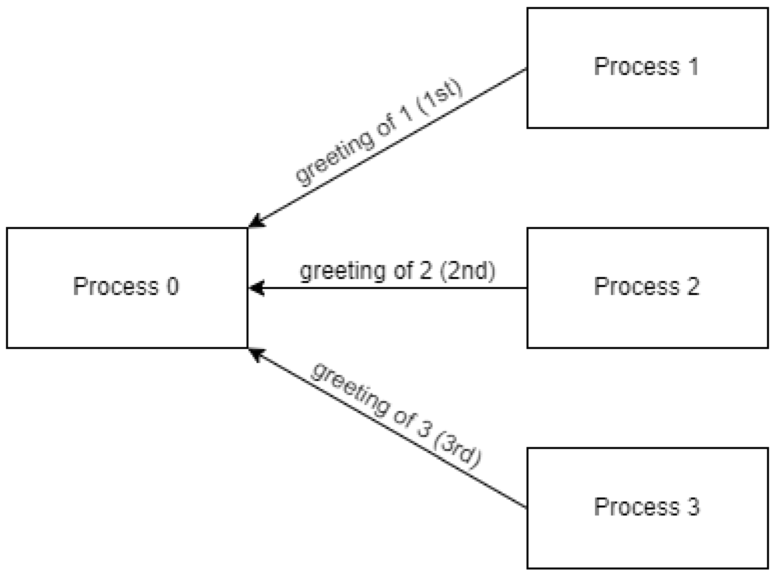
```
gbo@gbhoang:/mnt/d/2023-University-of-Adelaide/s1/PCD/pdc-23-s1-assign1$ mpiexec -n 1 ./hello
Greetings from process 0 of 1
gbo@gbhoang:/mnt/d/2023-University-of-Adelaide/s1/PCD/pdc-23-s1-assign1$ mpiexec -n 4 ./hello
Greetings from process 0 of 4
Greetings from process 1 of 4!
Greetings from process 2 of 4!
Greetings from process 3 of 4!
gbo@gbhoang:/mnt/d/2023-University-of-Adelaide/s1/PCD/pdc-23-s1-assign1$ mpiexec -n 4 ./hello
Greetings from process 0 of 4
Greetings from process 1 of 4!
Greetings from process 2 of 4!
Greetings from process 3 of 4!
```

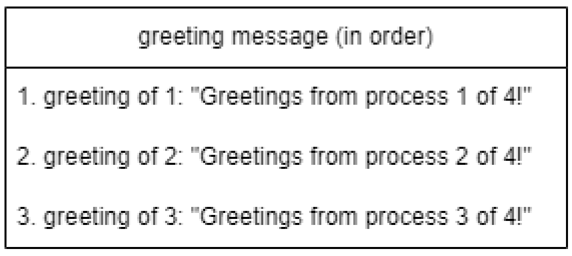
Figure 2: Outputs from the program hello.c. The first one is with one process. The next two are with four processes.

Step 2:

When the program is executed, the program will initialize the MPI environment. The number of processes based on the command ***mpiexec*** with the ***-n*** flag followed by the desired number of processes. The program will display the greeting messages from all the process, both the master (rank 0) and worker processes. In particular, if the process' rank is not 0, it will send its greeting message to the process rank 0 using ***MPI_Send()***. If the process' rank is 0, it will first print its own greeting message. Then it will receive message from all processes in incremental order (from rank 1 to rank n-1, where n is the number of processes) using ***MPI_Recv()*** in a for-loop. After all processes have finished their tasks, ***MPI_Finalize()*** is called to clean up the MPI environment.

Step 3:





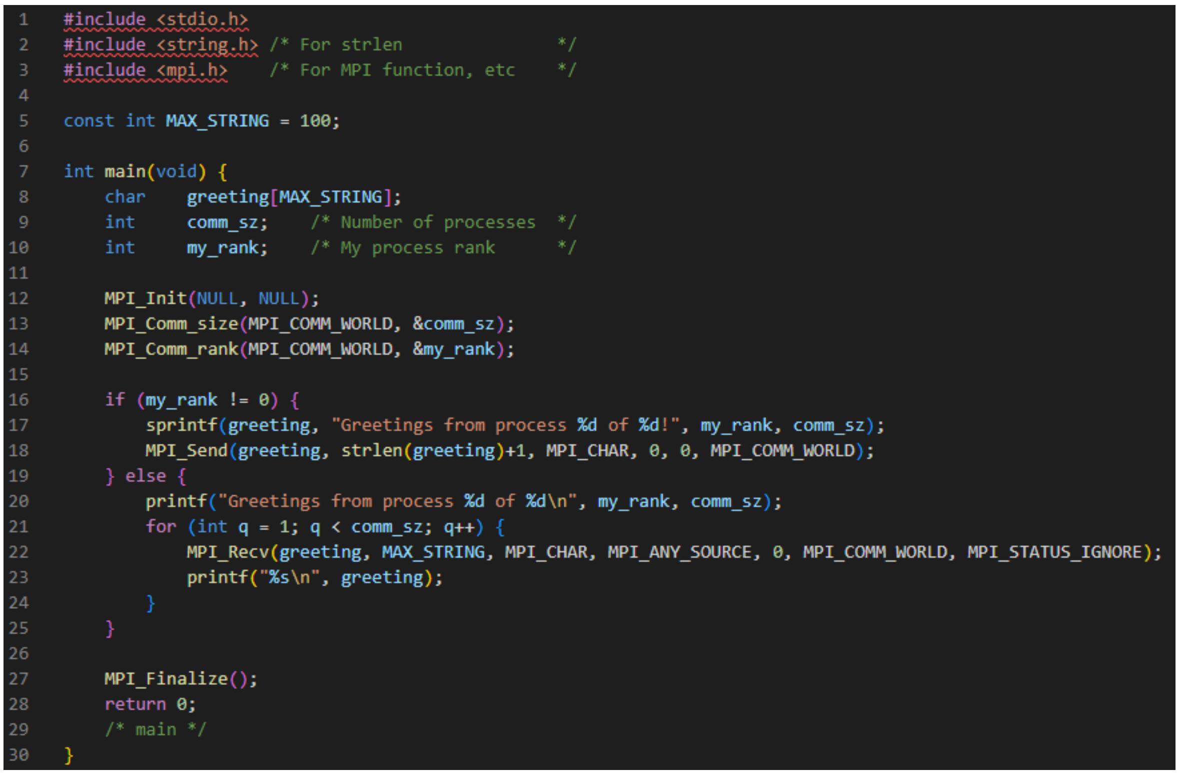
greeting message (in order)
1. greeting of 1: "Greetings from process 1 of 4!"
2. greeting of 2: "Greetings from process 2 of 4!"
3. greeting of 3: "Greetings from process 3 of 4!"

Figure 3: Diagram showing all the messages sent when executed hello.c with four processes. The greeting message are displayed in order.

Step 4:

In the program hello.c, the messages will be ordered in the same way for all executions. This is because the master process (rank 0) receives messages in a specific order based on the worker processes' ranks. The ***MPI_Recv()*** function is called in a loop, with the source parameter set to the loop counter variable **q**, which iterates through the ranks of the worker processes incrementally. This means that the master process always waits for messages in the same order, starting from rank 1 and proceeding incrementally until the last process.

Step 5:



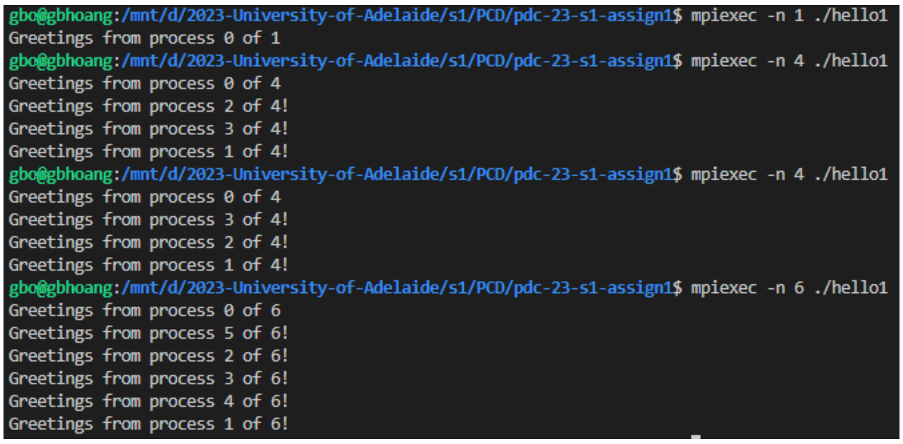
```c
#include <stdio.h>
#include <string.h> /* For strlen               */
#include <mpi.h>    /* For MPI function, etc    */

const int MAX_STRING = 100;

int main(void) {
    char    greeting[MAX_STRING];
    int     comm_sz;    /* Number of processes  */
    int     my_rank;    /* My process rank      */

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank != 0) {
        sprintf(greeting, "Greetings from process %d of %d!", my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    } else {
        printf("Greetings from process %d of %d\n", my_rank, comm_sz);
        for (int q = 1; q < comm_sz; q++) {
            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s\n", greeting);
        }
    }

    MPI_Finalize();
    return 0;
    /* main */
}
```

Figure 4: Program hello1.c

Step 6:

When the source parameter in the ***MPI_Recv()*** function call to ***MPI_ANY_SOURCE***, the order of the messages received may vary between executions. The change in the source parameter allows the master process can receive messages from any worker process, regardless of their rank as soon as they are ready to be received. Hence, the order in which the messages are received and displayed may not be the same for all executions.

Step 7:



```
gbo@gbhoang:/mnt/d/2023-University-of-Adelaide/s1/PCD/pdc-23-s1-assign1$ mpiexec -n 1 ./hello1
Greetings from process 0 of 1
gbo@gbhoang:/mnt/d/2023-University-of-Adelaide/s1/PCD/pdc-23-s1-assign1$ mpiexec -n 4 ./hello1
Greetings from process 0 of 4
Greetings from process 2 of 4!
Greetings from process 3 of 4!
Greetings from process 1 of 4!
gbo@gbhoang:/mnt/d/2023-University-of-Adelaide/s1/PCD/pdc-23-s1-assign1$ mpiexec -n 4 ./hello1
Greetings from process 0 of 4
Greetings from process 3 of 4!
Greetings from process 2 of 4!
Greetings from process 1 of 4!
gbo@gbhoang:/mnt/d/2023-University-of-Adelaide/s1/PCD/pdc-23-s1-assign1$ mpiexec -n 6 ./hello1
Greetings from process 0 of 6
Greetings from process 5 of 6!
Greetings from process 2 of 6!
Greetings from process 3 of 6!
Greetings from process 4 of 6!
Greetings from process 1 of 6!
```

Figure 5: Outputs from the program hello1.c. The first one is with one process. The next two are with four processes. The last one is with 6 processes.

To observe whether or not this happens, we can run the modified program multiple times with different number of processes:

1. Compile the modified program with an MPI compiler (**mpicc hello1.c -o hello1**).
2. Run the compiled program multiple times with different number of processes with the ***mpiexec*** command (in **Figure 5**, we use command **mpiexec -n 4 ./hello1** and **mpiexec -n 6 ./hello1**).

The program then will display the greeting message in different order between different executions. Notice that the order of the greeting messages can still be the same on some executions.