Assignment 1 Final Report

Part 1:

I tried implement my final design used the ring.c resource uploaded in myuni. I used this as my template because it already has many of the features implemented. I just needed to make some modifications and write the printf functions at the correct spot as they are vital in ensuring that the code is correctly outputted. My program is missing the reading in the random m-value specification because I tried it that way first but ran into errors. Some takeaways from my initial attempt were: defining int rank, value, size, false=0; int right_nbr, left_nbr; int arr[10],arr2[10],i, n; MPI_Comm ring_comm; MPI_Status status;.

Since I didn't use the m-values method, I used a for loop and an if statement in initialising all token values for each of the processes. I found this to be an efficient method as we just have to check that the current process is 0, which is also when the current rank is 0. If this condition is met, then every token is initialised to the value of the process. For example, process 0 begins with token 0, process 1 begins with token 1, process 2 begins with token 2, and process 3 begins with token 3. I also used the function MPI_Cart_create which makes a new communicator to which topology information has been attached. Along with the MPI_Cart_shift function, which is used to find two "nearby" neighbours of the calling process along a specific direction of an N-dimensional cartesian topology.

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
    int rank, size, value = 0, reorder = 0;
    int right_n, left_n;

    MPI_Comm ring;
    MPI_Status status;

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD , &size);
    MPI_Cart_create(MPI_COMM_WORLD, 1, &size, &reorder, 1, &ring);
    MPI_Cart_shift(ring, 0, 1, &left_n, &right_n);
    MPI_Comm_rank(ring, &rank);

   for(int i = 0; i < 4; ++i) {
        if(rank == 0) {
            value = i;
            MPI_Send(&value, 1, MPI_INT, right_n, 0 , ring);
            printf("Process %d sending token %d to process 1\n", rank, value);
        }
        else {
            MPI_Recv(&value, 1, MPI_INT, left_n, 0 , ring, &status );

            if(rank < size - 1)
                MPI_Send(&value, 1, MPI_INT, right_n, 0 , ring);
                    printf("Process %d sending token %d to process %d\n", rank, value, rank + 1);
        }

        printf("Process %d of %d received %d\n", rank, size - 1, value);
    }
    MPI_Finalize();
    return 0;
}
```

Output:

Process 0 sending token 0 to process 1
Process 0 of 3 received 0
Process 0 sending token 1 to process 1
Process 0 of 3 received 1
Process 0 sending token 2 to process 1

Process 0 of 3 received 2
Process 0 sending token 3 to process 1
Process 0 of 3 received 3
Process 1 sending token 0 to process 2
Process 1 of 3 received 0
Process 1 sending token 1 to process 2
Process 1 of 3 received 1
Process 1 sending token 2 to process 2
Process 1 of 3 received 2
Process 1 sending token 3 to process 2
Process 1 of 3 received 3
Process 2 sending token 0 to process 3
Process 2 of 3 received 0
Process 2 sending token 1 to process 3
Process 2 of 3 received 1
Process 2 sending token 2 to process 3
Process 2 of 3 received 2
Process 2 sending token 3 to process 3
Process 2 of 3 received 3
Process 3 sending token 0 to process 4
Process 3 of 3 received 0
Process 3 sending token 1 to process 4
Process 3 of 3 received 1
Process 3 sending token 2 to process 4
Process 3 of 3 received 2
Process 3 sending token 3 to process 4
Process 3 of 3 received 3

However, the output for this gave me issues. The process doesn't change and the outputs are just looped not actually shifted. Since, this method was also not working I used a Circular Shift template:

```c
#include <stdio.h>
#include <mpi.h>
#define MSIZE 500

int main(int argc, char *argv[])
{
    MPI_Status status;
    int rank, size, tag, to, from;

    int i;
    int A[MSIZE], B[MSIZE];

    /* Start up MPI */
/* ---> INSERT MPI code */

    /* Arbitrarily choose tag.  Calculate the rank of the to process in the ring.
    Use the modulus operator so that the to process in the ring.  Use the modulus operator so that the
       last process "wraps around" to rank zero. */

    tag = 201;
    to = (rank + 1) % size;
    from = (rank + size - 1) % size;

    for (i = 0; i < MSIZE; i++)
        A[i] = rank;
    /* sendrecv of array A */
/* ---> INSERT MPI code */

    printf("Process %d sending %d to %d\n",
        rank, MSIZE, to);

    printf("Process %d received %d\n",
        rank, MSIZE, from);
    /* print first content of arrays A and B */
    printf("Proc %d has A[0] = %d, B[0] = %d\n\n", rank, A[0], B[0]);

    MPI_Finalize();
    return 0;
}
```

Using this template, I was able to make a circular ring program that will ask for however many shifts and shifts into chronological form. So, if the output will be in the form of: 0 -> 1 -> 2 -> 3. This strategy is basically reverse style of implemented and it will be clearly explained in part 2. I wanted to try a different style because I was having errors with the original specifications.

Part 2:

The strategy I used for getting the output was with neighbour relationship table and diagrams. The table and diagrams check the left and right neighbours after every shift. This also allowed me to model an expected outcome.

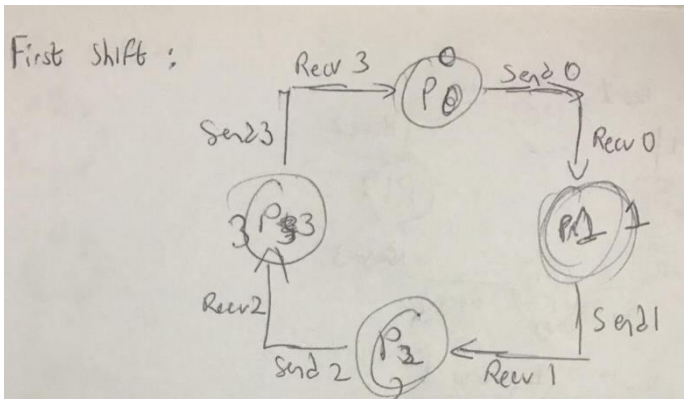
Non-circular/ring table:

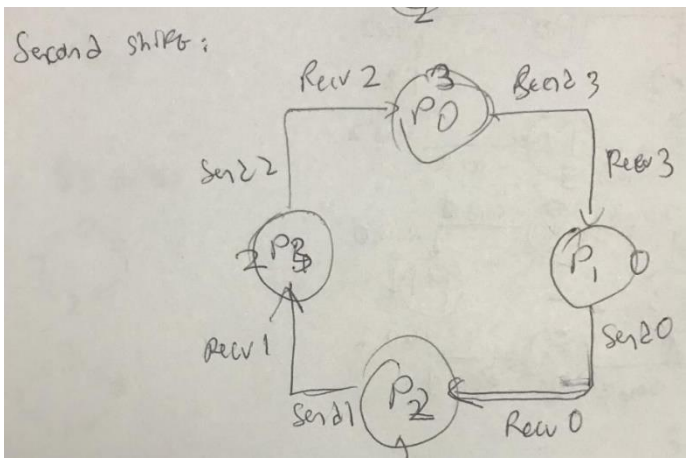
	Process 0	Process 1	Process 2	Process 3
Left (send)	-	0	1	2
Right (receive)	1	2	3	-

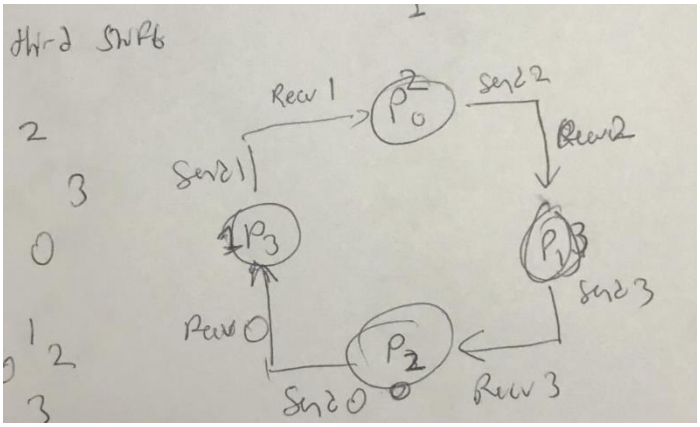
Circular/ring table:

	Process 0	Process 1	Process 2	Process 3
Left (send)	3	0	1	2
Right (receive)	1	2	3	0

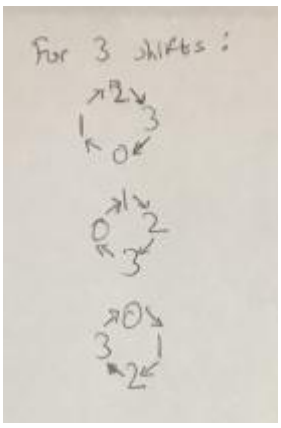
With this logic, I did a diagram of processes after 3 shifts.

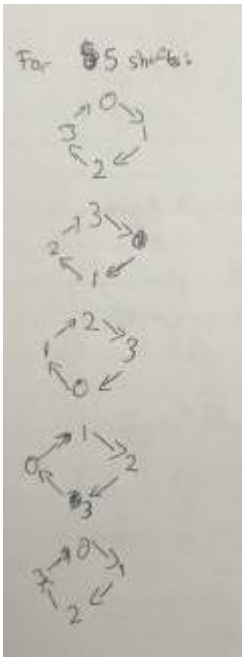


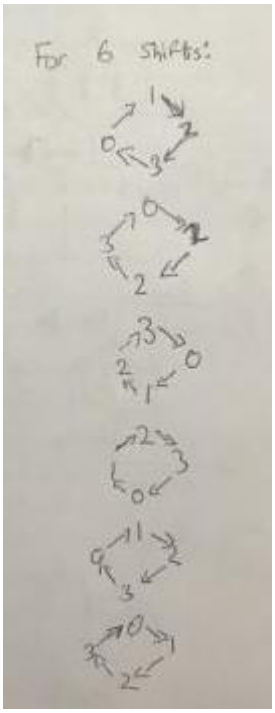


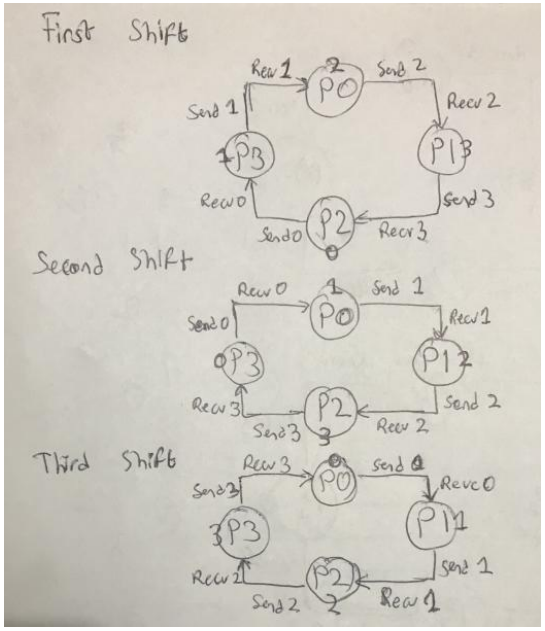


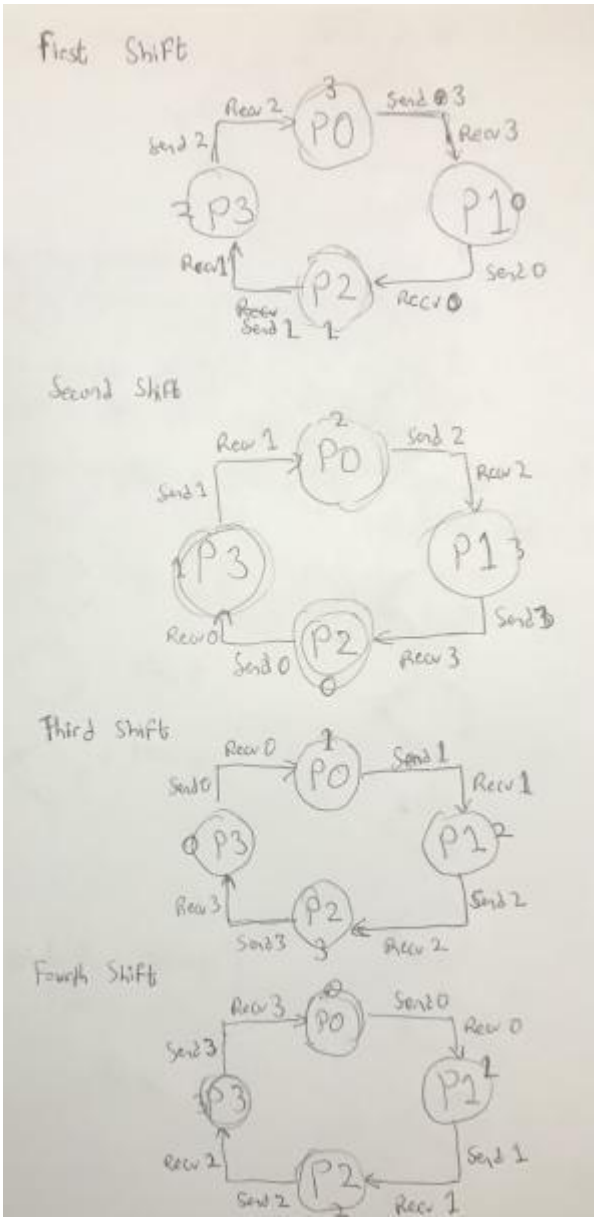
Due to the errors from previous attempts, in this program I built it in a reverse structure. In the program we start off with the shifted form and then shift it to its original chronological form. I first planned this on paper for multiple different shifts. For example, if its 3 shifts I want to start at the shifted form 2->3->0->1 and end at the result 0->1->2->3. For 4 shifts, program starts at 3->0->1->2 (shifted form) and end at result 0->1->2->3. Below is illustrations of this method.











With the help of the table and diagrams I was able to come up with an expected output after three shifts.

Expected output:

Process 0 sending 2 to 1 (right)
Process 1 received 2 (left)
Process 1 sending 2 to 2 (right)
Process 2 received 2 (left)
Process 2 sending 2 to 3 (right)
Process 3 received 2 (left)
Process 3 sending 2 to 0 (right)
Process 0 received 2 (left)
Process 0 sending 1 to 1 (right)
Process 1 received 1 (left)
Process 1 sending 1 to 2 (right)

Process 2 received 1 (left)
Process 2 sending 1 to 3 (right)
Process 3 received 1 (left)
Process 3 sending 1 to 0 (right)
Process 0 received 1 (left)
Process 0 sending 0 to 1 (right)
Process 1 received 0 (left)
Process 1 sending 0 to 2 (right)
Process 2 received 0 (left)
Process 2 sending 0 to 3 (right)
Process 3 received 0 (left)
Process 3 sending 0 to 0 (right)

Part 3:

Extract of Output:

Enter the number of times around the ring: 3
Process 0 sending 2 to 1 (right)
Process 1 received 2 (left)
Process 1 sending 2 to 2 (right)
Process 2 received 2 (left)
Process 2 sending 2 to 3 (right)
Process 3 received 2 (left)
Process 3 sending 2 to 0 (right)
Process 0 received 2 (left)
Process 0 sending 1 to 1 (right)
Process 1 received 1 (left)
Process 1 sending 1 to 2 (right)
Process 2 received 1 (left)
Process 2 sending 1 to 3 (right)
Process 3 received 1 (left)
Process 3 sending 1 to 0 (right)
Process 0 received 1 (left)
Process 0 sending 0 to 1 (right)
Process 1 received 0 (left)
Process 1 sending 0 to 2 (right)
Process 2 received 0 (left)
Process 2 sending 0 to 3 (right)
Process 3 received 0 (left)
Process 3 sending 0 to 0 (right)

Image:

assignment1.c hello.c

```c
// Simple circular ring program that will ask for how many shifts and shifts into

#include <stdio.h>
#include <mpi.h> // For MPI functions, etc

int main(int argc, char *argv[]) {
  // Initialising; rank = process rank, size = number of process
  MPI_Status status;
  int a, rank, size, tag, to, from;

  // Initialize the MPI environment
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  tag = 1; // Randomly choosen.
  // Messages are sent with an accompanying user-defined integer tag,
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1 COMMENTS

```
ubuntu@codespaces_861a96:/workspaces/65145297$ mpicc assignment1.c -o mpi_hello -std=c99
ubuntu@codespaces_861a96:/workspaces/65145297$ mpiexec -n 4 ./mpi_hello
Enter the number of times around the ring: 3
Process 0 sending 2 to 1 (right)
Process 1 received 2 (left)
Process 1 sending 2 to 2 (right)
Process 2 received 2 (left)
Process 2 sending 2 to 3 (right)
Process 3 received 2 (left)
Process 3 sending 2 to 0 (right)
Process 0 received 2 (left)
Process 0 sending 1 to 1 (right)
Process 1 received 1 (left)
Process 1 sending 1 to 2 (right)
Process 2 received 1 (left)
Process 2 sending 1 to 3 (right)
Process 3 received 1 (left)
Process 3 sending 1 to 0 (right)
Process 0 received 1 (left)
Process 0 sending 0 to 1 (right)
Process 1 received 0 (left)
Process 1 sending 0 to 2 (right)
Process 2 received 0 (left)
Process 2 sending 0 to 3 (right)
Process 3 received 0 (left)
Process 3 sending 0 to 0 (right)
```

Part 4:

The program works for multiple shifts and multiple processes.

For 6 processes and 3 shifts:

```
ubuntu@codespaces_861a96:/workspaces/65145297$ ^C
ubuntu@codespaces_861a96:/workspaces/65145297$ mpiexec -n 6 ./mpi_hello
Enter the number of times around the ring: 3
Process 0 sending 2 to 1 (right)
Process 1 received 2 (left)
Process 1 sending 2 to 2 (right)
Process 2 received 2 (left)
Process 2 sending 2 to 3 (right)
Process 3 received 2 (left)
Process 3 sending 2 to 4 (right)
Process 4 received 2 (left)
Process 4 sending 2 to 5 (right)
Process 5 received 2 (left)
```

For 4 processes and 5 shifts:

```
ubuntu@codespaces_861a96:/workspaces/65145297$ mpiexec -n 4 ./mpi_hello
Enter the number of times around the ring: 5
Process 0 sending 4 to 1 (right)
Process 1 received 4 (left)
Process 1 sending 4 to 2 (right)
Process 2 received 4 (left)
Process 2 sending 4 to 3 (right)
Process 3 received 4 (left)
Process 3 sending 4 to 0 (right)
Process 0 received 4 (left)
Process 0 sending 3 to 1 (right)
Process 1 received 3 (left)
Process 1 sending 3 to 2 (right)
Process 2 received 3 (left)
Process 2 sending 3 to 3 (right)
Process 3 received 3 (left)
Process 3 sending 3 to 0 (right)
```