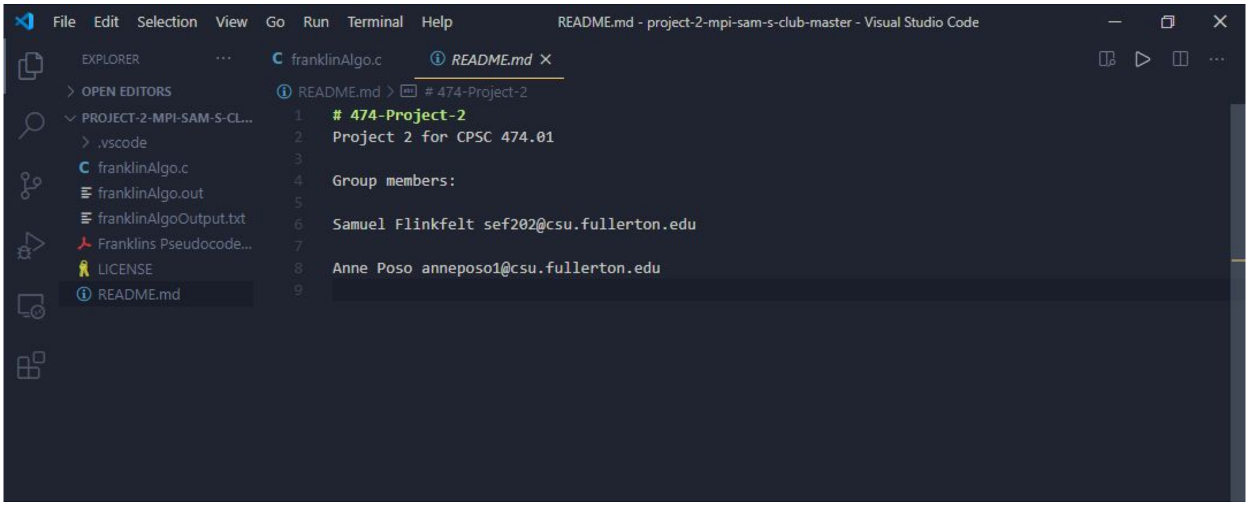
Anne Poso anneposo1@csu.fullerton.edu
Samuel Flinkfelt sef202@csu.fullerton.edu
474 Distributed Computing Project 2
Franklin's Algorithm

The topic that we chose was "Write a distributed program in MPI that simulates one of the leader election algorithms learnt in class for a ring topology using a fixed number of processors." We chose to implement Franklin's Algorithm in the C programming language. In this report we go over the pseudocode of our implementation of Non-blocking MPI programming of Franklin's Algorithm and describe how to compile and run the program. At the end of the report we provide sample runs of the program with 2 and 4 processors.



```
# 474-Project-2
Project 2 for CPSC 474.01

Group members:

Samuel Flinkfelt sef202@csu.fullerton.edu

Anne Poso anneposo1@csu.fullerton.edu
```

Pseudocode

```
Main{
        int rank, size
        initialize MPI // Initializes MPI
        initialize Comm Size // Figures out the number of processors I am asking for
        initalize Rank // Figures out which rank we are
        bool active_array = {true, true, true, ,true} // if a process is active or passive  bool
        is_Elected = false; // if a process has been elected leader

        // multiple election rounds until one node elected as leader
        while(not is_Elected)
        out_message1 = outMessage2 = rankID[rank]
        print process ID and rank
        if( active_array[rank] )
                if (rank == 0)
                        // send rankID msg to both neighbor nodes
                        send MPI message neighbor right
                        send MPI message to neighbor left
                        // receive rankID msgs from both neighbor nodes
                        receive message from right neighbor
```

```
            receive message from left neighbor
        else if (rank == size -1)
            send MPI message neighbor right
            send MPI message to neighbor left
            receive message from right neighbor
            receive message from left neighbor
        else
            send MPI message neighbor right
            send MPI message to neighbor left
            receive message from right neighbor
            receive message from left neighbor

        // wait for receive operations to complete to guarantee message delivery
        MPI wait receive right neighbor
        MPI wait receive left neighbor

        // if current node is lesser than any of its neighbor nodes
        If (rightmsg > rankID[rank] OR leftmsg > rankID[rank])
            Is_Active = false
            print process has become active
        // if receiving msgs are same ID as current node,
        // then it is the last active node, so it is elected as leader
        else if (rightmsg is equal rankID[rank] OR leftmsg is equal rankID[rank])
            is_Elected = true
            print process has become leader

else //process is passive
    if rank == 0
        // receive rankID msgs from neighbor nodes
        receive message from right neighbor
        receive message from left neighbor
        // forward the same msg buffer to next node
        send message to right neighbor
        send message to left neighbor
    else if rank == size - 1
        receive message from right neighbor
        receive message from left neighbor
        send message to right neighbor
        send message to left neighbor


    else
        receive message from right neighbor
        receive message from left neighbor
        send message to right neighbor
        send message to left neighbor
    // wait for receive operations to complete to guarantee message delivery
    MPI wait receive right neighbor
    MPI wait receive left neighbor

MPI_Abort // terminate processes after leader is elected
//MPI_Finalize // shutdown MPI
return 0
```

<u>How To Run Code:</u>

We installed OpenMPI on our machines with:

```
sudo apt install openmpi-bin libopenmpi-dev
```

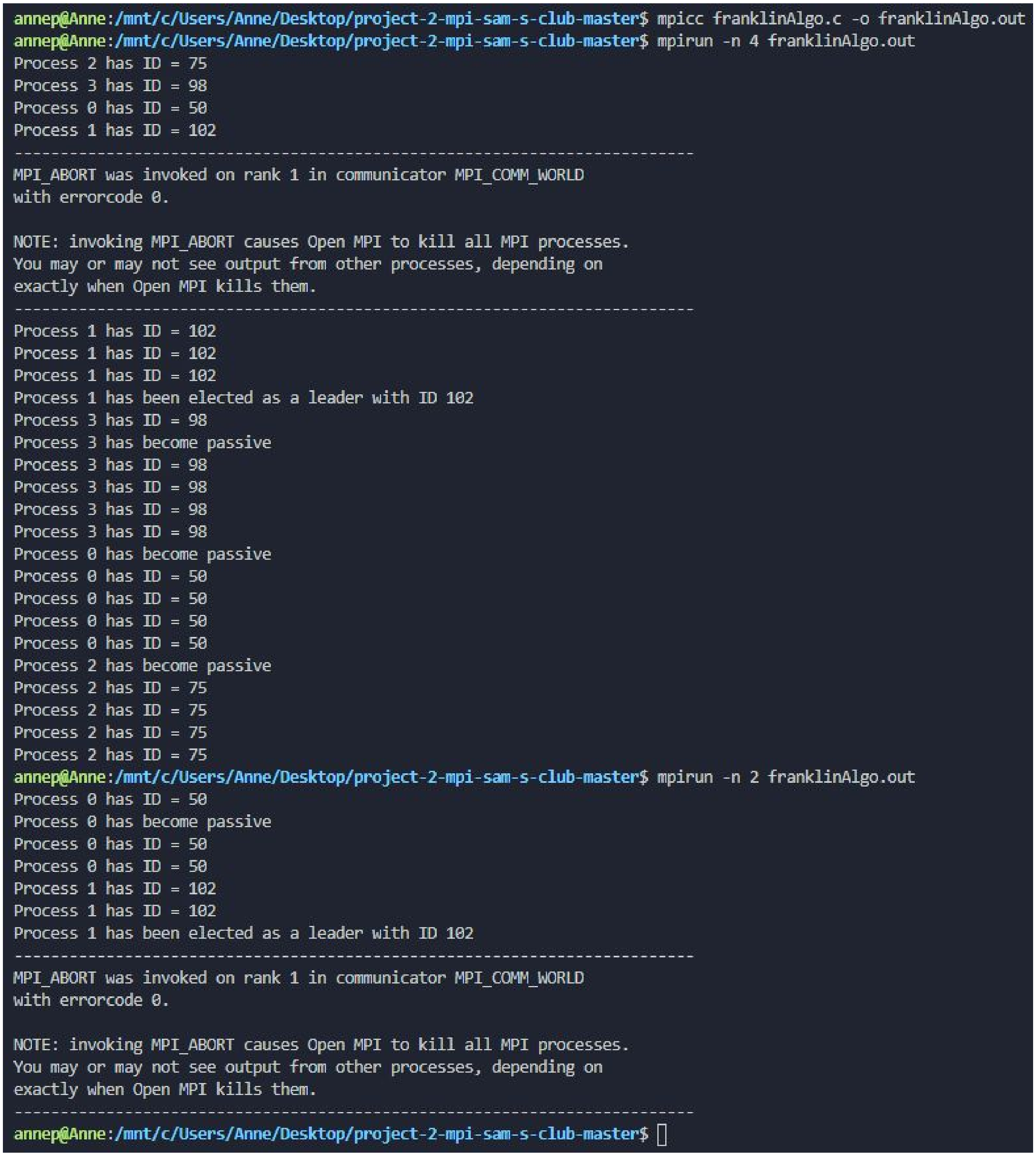
Compile with:

```
mpicc franklinAlgo.c -o franklinAlgo.out
```

Run with X processors:

```
mpirun -n X franklinAlgo.out
```

Sample run with 4 processors and 2 processors:



```
annep@Anne:/mnt/c/Users/Anne/Desktop/project-2-mpi-sam-s-club-master$ mpicc franklinAlgo.c -o franklinAlgo.out
annep@Anne:/mnt/c/Users/Anne/Desktop/project-2-mpi-sam-s-club-master$ mpirun -n 4 franklinAlgo.out
Process 2 has ID = 75
Process 3 has ID = 98
Process 0 has ID = 50
Process 1 has ID = 102
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 1 in communicator MPI_COMM_WORLD
with errorcode 0.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--------------------------------------------------------------------------
Process 1 has ID = 102
Process 1 has ID = 102
Process 1 has ID = 102
Process 1 has been elected as a leader with ID 102
Process 3 has ID = 98
Process 3 has become passive
Process 3 has ID = 98
Process 3 has ID = 98
Process 3 has ID = 98
Process 3 has ID = 98
Process 0 has become passive
Process 0 has ID = 50
Process 0 has ID = 50
Process 0 has ID = 50
Process 0 has ID = 50
Process 2 has become passive
Process 2 has ID = 75
Process 2 has ID = 75
Process 2 has ID = 75
Process 2 has ID = 75
annep@Anne:/mnt/c/Users/Anne/Desktop/project-2-mpi-sam-s-club-master$ mpirun -n 2 franklinAlgo.out
Process 0 has ID = 50
Process 0 has become passive
Process 0 has ID = 50
Process 0 has ID = 50
Process 1 has ID = 102
Process 1 has ID = 102
Process 1 has been elected as a leader with ID 102
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 1 in communicator MPI_COMM_WORLD
with errorcode 0.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--------------------------------------------------------------------------
annep@Anne:/mnt/c/Users/Anne/Desktop/project-2-mpi-sam-s-club-master$ 
```