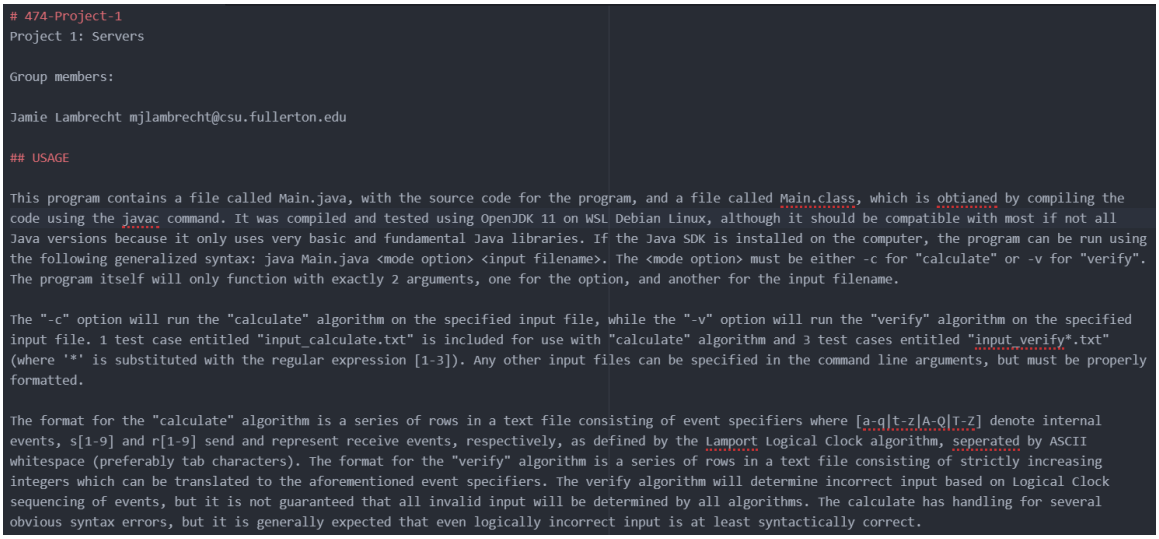
# CPSC 474-01 Project 1

Jamie Lambrecht
mjlambrecht@csu.fullerton.edu

October 14, 2021

# 1 Screenshot of names in README.md



```
# 474-Project-1
Project 1: Servers

Group members:

Jamie Lambrecht mjlambrecht@csu.fullerton.edu

## USAGE

This program contains a file called Main.java, with the source code for the program, and a file called Main.class, which is obtianed by compiling the
code using the javac command. It was compiled and tested using OpenJDK 11 on WSL Debian Linux, although it should be compatible with most if not all
Java versions because it only uses very basic and fundamental Java libraries. If the Java SDK is installed on the computer, the program can be run using
the following generalized syntax: java Main.java <mode option> <input filename>. The <mode option> must be either -c for "calculate" or -v for "verify".
The program itself will only function with exactly 2 arguments, one for the option, and another for the input filename.

The "-c" option will run the "calculate" algorithm on the specified input file, while the "-v" option will run the "verify" algorithm on the specified
input file. 1 test case entitled "input_calculate.txt" is included for use with "calculate" algorithm and 3 test cases entitled "input_verify*.txt"
(where '*' is substituted with the regular expression [1-3]). Any other input files can be specified in the command line arguments, but must be properly
formatted.

The format for the "calculate" algorithm is a series of rows in a text file consisting of event specifiers where [a-q|t-z|A-Q|T-Z] denote internal
events, s[1-9] and r[1-9] send and represent receive events, respectively, as defined by the Lamport Logical Clock algorithm, seperated by ASCII
whitespace (preferably tab characters). The format for the "verify" algorithm is a series of rows in a text file consisting of strictly increasing
integers which can be translated to the aforementioned event specifiers. The verify algorithm will determine incorrect input based on Logical Clock
sequencing of events, but it is not guaranteed that all invalid input will be determined by all algorithms. The calculate has handling for several
obvious syntax errors, but it is generally expected that even logically incorrect input is at least syntactically correct.
```

# 2 Usage

This program contains a file called Main.java, with the source code for the program, and a file called Main.class, which is obtianed by compiling the code using the javac command. It was compiled and tested using OpenJDK 11 on WSL Debian Linux, although it should be compatible with most if not all Java versions because it only uses very basic and fundamental Java libraries. If the Java SDK is installed on the computer, the program can be run using the following generalized syntax: java Main.java ⟨mode option⟩ ⟨input filename⟩. The ⟨mode option⟩ must be either -c for "calculate" or -v for "verify". The program itself will only function with exactly 2 arguments, one for the option, and another for the input filename.

The "-c" option will run the "calculate" algorithm on the specified input file, while the "-v" option will run the "verify" algorithm on the specified input file. 1 test case entitled "input_calculate.txt" is included for use with "calculate" algorithm and 3 test cases entitled "input_verify*.txt" (where '*' is substituted with the regular expression [1-3]). Any other input files can be specified in the command line arguments, but must be properly formatted.

The format for the "calculate" algorithm is a series of rows in a text file consisting of event specifiers where [a-q—t-z—A-Q—T-Z] denote internal events, s[1-9] and r[1-9] send and represent receive events, respectively, as defined by the Lamport Logical Clock algorithm, seperated by ASCII whitespace (preferably tab characters). The format for the "verify" algorithm is a series of rows in a text file consisting of strictly increasing integers which can be translated to the aforementioned event specifiers. The verify algorithm will determine incorrect input based on Logical Clock sequencing of events, but it is not guaranteed that all invalid input will be determined by all algorithms. The calculate has handling for several obvious syntax errors, but it is generally expected that even logically incorrect input is at least syntactically correct.

# 3 Psuedocode

## 3.1 "Calculate" Algorithm

```
while not all rows are complete:
  if the currently indexed row is not complete:
    if the row does not have a waiting receive event:
      if the current row's index is beyond the row bounds:
        add the row to the set of completed rows
      if we see a null element:
        set the corresponding element in the output matrix to 0
        increment the index for the row
      else if the current element begins with ''r'':
        if the element has 2 chars:
          if the second character is a digit:
            set the waiting_for for the row to that digit
      else if the current element begins with ''s'':
        if the element has 2 chars:
          if the second character is a digit:
            If we have not seen a send for that digit:
              record the logical clock value for the send
              write the LC to the output and increment the LC for the row
              increment the current index for the current row
            else if the current element has only one char:
        if the current element's char is a letter:
          write the current LC to output and incremenet LC for the row
          increment the current index for the current row
      else:
        if a send has happened that corresponds with the receive:
          record output as max of (LC for the row, send value for the receive + 1)
          set the LC for the row to the this output + 1
          set the waiting_for for the row back to 0
          remove the this receive from the set of sends
        else:
          go the next row
  else:
    go to the next row

return the completed output
```

## 3.2 "Verify" Algorithm

```
while not all rows are complete:
  if we have not seen all rows at least once:
    if the row is the last row:
      we have seen all the rows at least once (set boolean flag)
  if the currently indexed row is not complete:
    if the current index is not less than the number of elements in a row:
      add current row to set of completed rows, restart the loop
    if the current element has been recorded as null:
      loop thru the rest of the elements, recording as null until the row is complete
      restart the loop
    if the row has been marked with an ''s'' (either send or internal):
      if the element precedes a the LC of a pending receive event
        record the sequential number of the send event
        change the value for the receive LC in the map from 0 to the send number
        increment the current row's current index
        record the current index as the first_undetermined for the row
        record the current element as the last determined element for the row
      else:
        if there are no pending recieve LC numbers in the map
        or the current element is not less than the maximum send number associated with a receive event:
          ensure that the element is sequential to the last determined element for the row
          record and increment the letter enumerator for internal events to the output
          increment the current index for the row
          record the current index as the first_undetermined for the row
          record the current element as the last determined element for the row
        else:
          if there are no recieve LCs in the map (completely resolved)
          or element is not less than the maximum known send number corresponding to a receive LC:
            ensure that the element is sequential to the last determined element for the row
            record and increment the letter enumerator for internal events to the output
            increment the current index for the row
            record the current index as the first_undetermined for the row
            record the current element as the last determined element for the row
          else if all other rows are waiting on sends and we have cycled thru all rows at least once:
            ensure that the element is sequential to the last determined element for the row
            record and increment the letter enumerator for internal events to the output
            increment the current index for the row
            record the current index as the first_undetermined for the row
            record the current element as the last determined element for the row
          else:
            go to the next row
    if the row has been marked with an ''r'' (receive):
      ensure that the element is sequential to the last determined element for the row
```

```
      if all other rows are waiting on sends
      and the element is greater than the max LC seen and the element is the minimum receive LC:
        the sequence is incorrect, return an error
    if the recieve LC has a known corresponding send (value is not 0):
      append the value to the output
      set the expected LC for the row to the element + 1
      increment the current index for the row
      record the current index as the first_undetermined for the row
      record the current element as the last determined element for the row
      remove the element from the receive LCs map
      decrement the ''pending receives'' count
    else:
      go to the next row
  else:
    if the current element in a row matches a sequential (expected) LC value:
      mark the output with ''s'' (internal or send)
      keep track of the largest sequential event number both by row and globally
      if first_undetermined (index) for the row is not set or has been reset (-1):
        set that value to the current index for the row
      if the current index is before the last index for the row:
        increment the current index for the row
        increment the expected LC for the row
      else:
        set the current index for the row back to the first undetermined for the row
    else if the element is non-zero:
      mark the output with ''r'' (receive)
      if first_undetermined (index) for the row is not set or has been reset (-1):
        set that value to the current index for the row to 0
      else:
        set the current index for the row to the first undetermined for the row
      increment the count of ''pending receives''
      reset the first_undetermined for the row (-1)
    else:
      set the output for the current position to null
      if the current index for the row is not the first undetermined for the row
        set the currrent for the row to the first undetermined for the row
      else:
        increment the current index for the row
else:
  go to the next row
```

# 4 Screenshot of program output

## 4.1 Calculate

```
jamie@DESKTOP-MB7TS2J:/mnt/c/Users/mjlam/School/Current/CPSC474/Project 1/project-1-servers-jamie-lambrecht$ java Main.java -c input_calculate.txt
1       2       8       9
1       6       7       0
3       4       5       6
```

## 4.2 Verify (Test case 1)

```
jamie@DESKTOP-MB7TS2J:/mnt/c/Users/mjlam/School/Current/CPSC474/Project 1/project-1-servers-jamie-lambrecht$ java Main.java -v input_verify1.txt
a       s1      r3      e
b       r2      s3      null
r1      c       s2      d
```

## 4.3 Verify (Test case 2)

```
jamie@DESKTOP-MB7TS2J:/mnt/c/Users/mjlam/School/Current/CPSC474/Project 1/project-1-servers-jamie-lambrecht$ java Main.java -v input_verify2.txt
s1      a       r3      e
b       r2      s3      null
r1      c       d       s2
```

## 4.4 Verify (Test case 3)

```
jamie@DESKTOP-MB7TS2J:/mnt/c/Users/mjlam/School/Current/CPSC474/Project 1/project-1-servers-jamie-lambrecht$ java Main.java -v input_verify3.txt
INCORRECT
```