Michael Moon | m-moon@csu.fullerton.edu

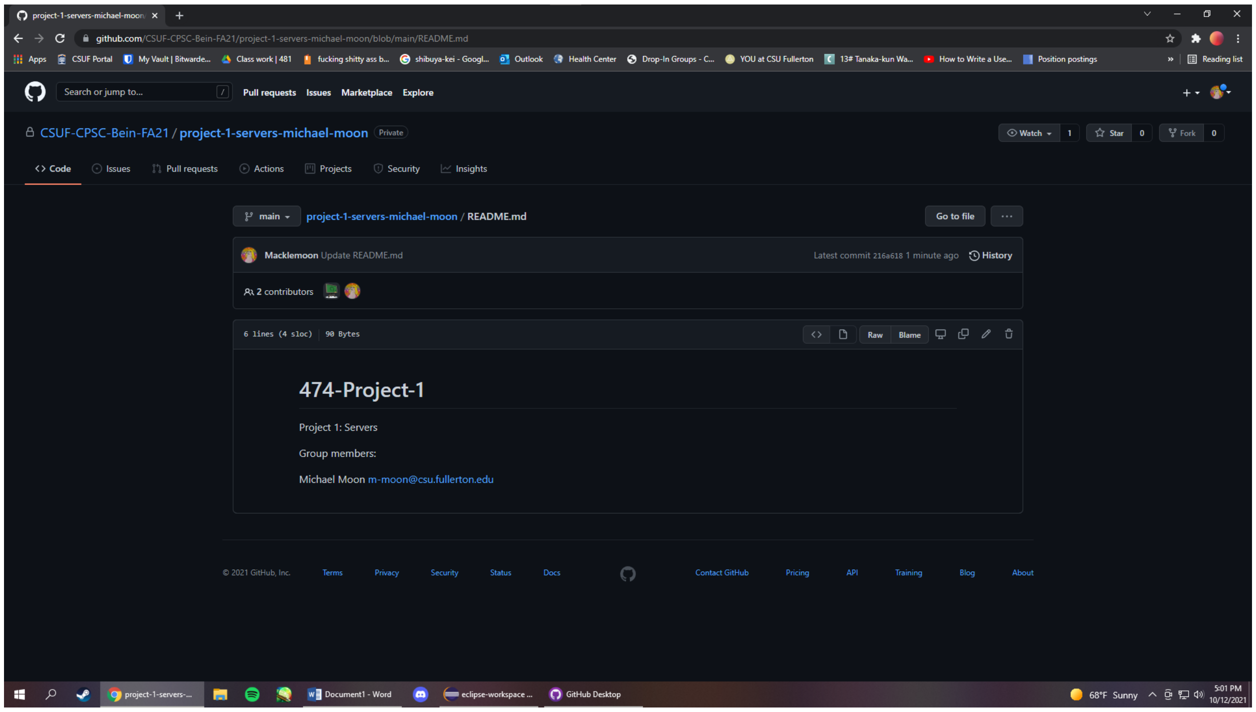
CPSC 474

Dr. Bein

15 October 2021

Project 1

Full screenshot of team name(s), email address(es) and proof of work on Project 1.



CSUF-CPSC-Bein-FA21 / project-1-servers-michael-moon

main project-1-servers-michael-moon / README.md

Macklemoon Update README.md — Latest commit 216a618 1 minute ago

2 contributors

6 lines (4 sloc) | 90 Bytes

474-Project-1

Project 1: Servers

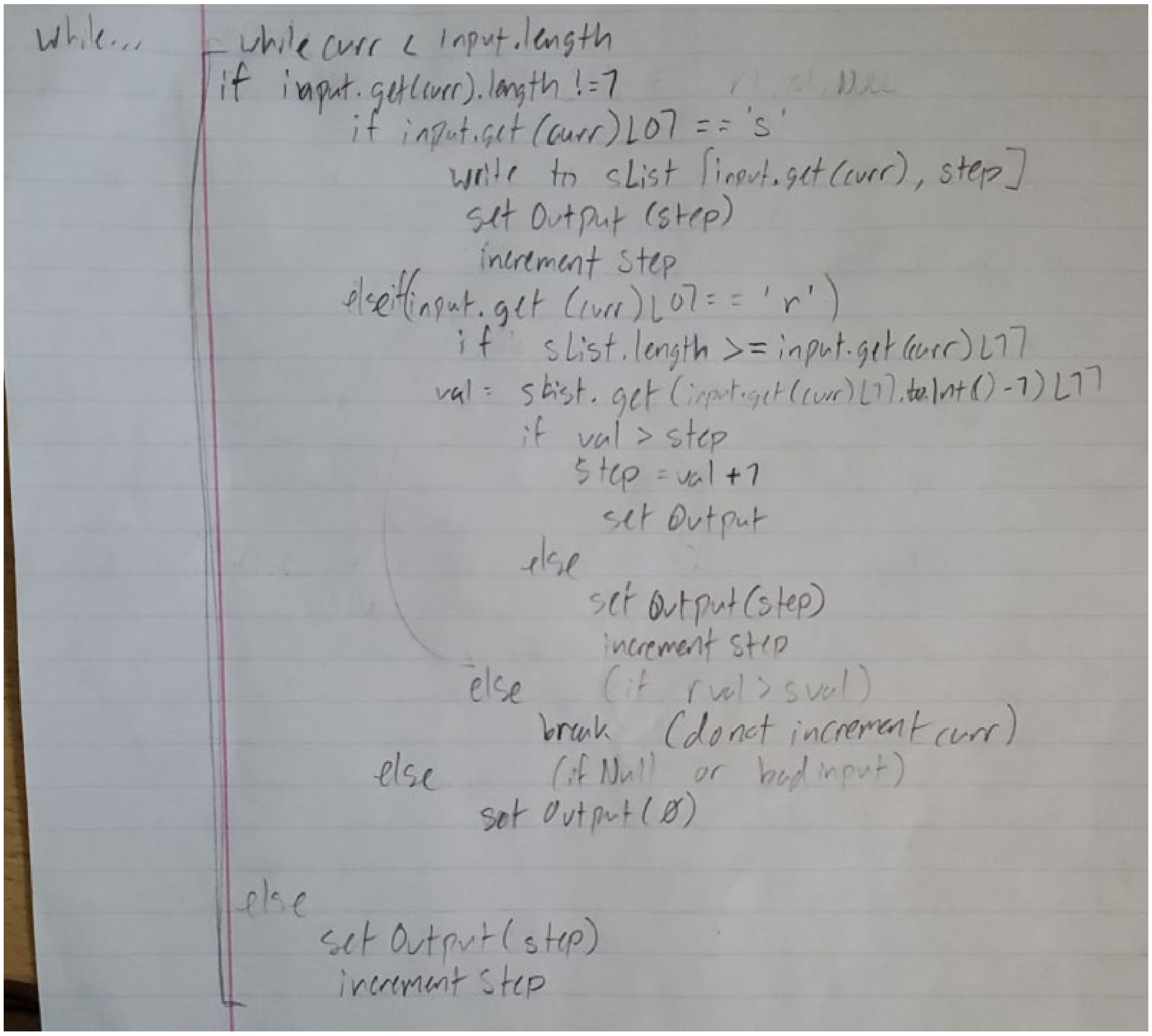
Group members:

Michael Moon m-moon@csu.fullerton.edu

Pseudocode

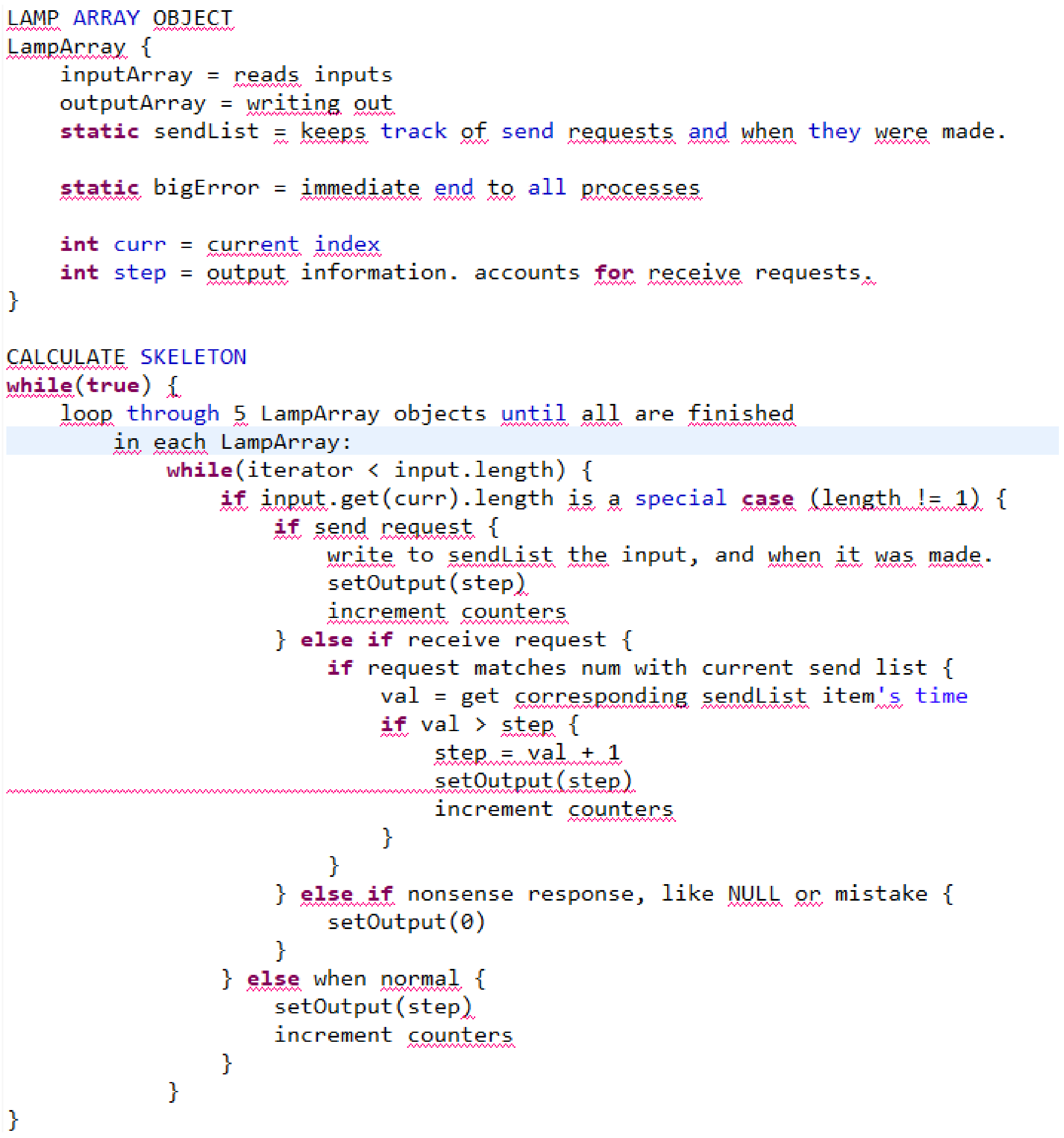
Screenshots of pseudocode used in project development, and typed alternatives are provided:

Pseudocode used for Calculate().



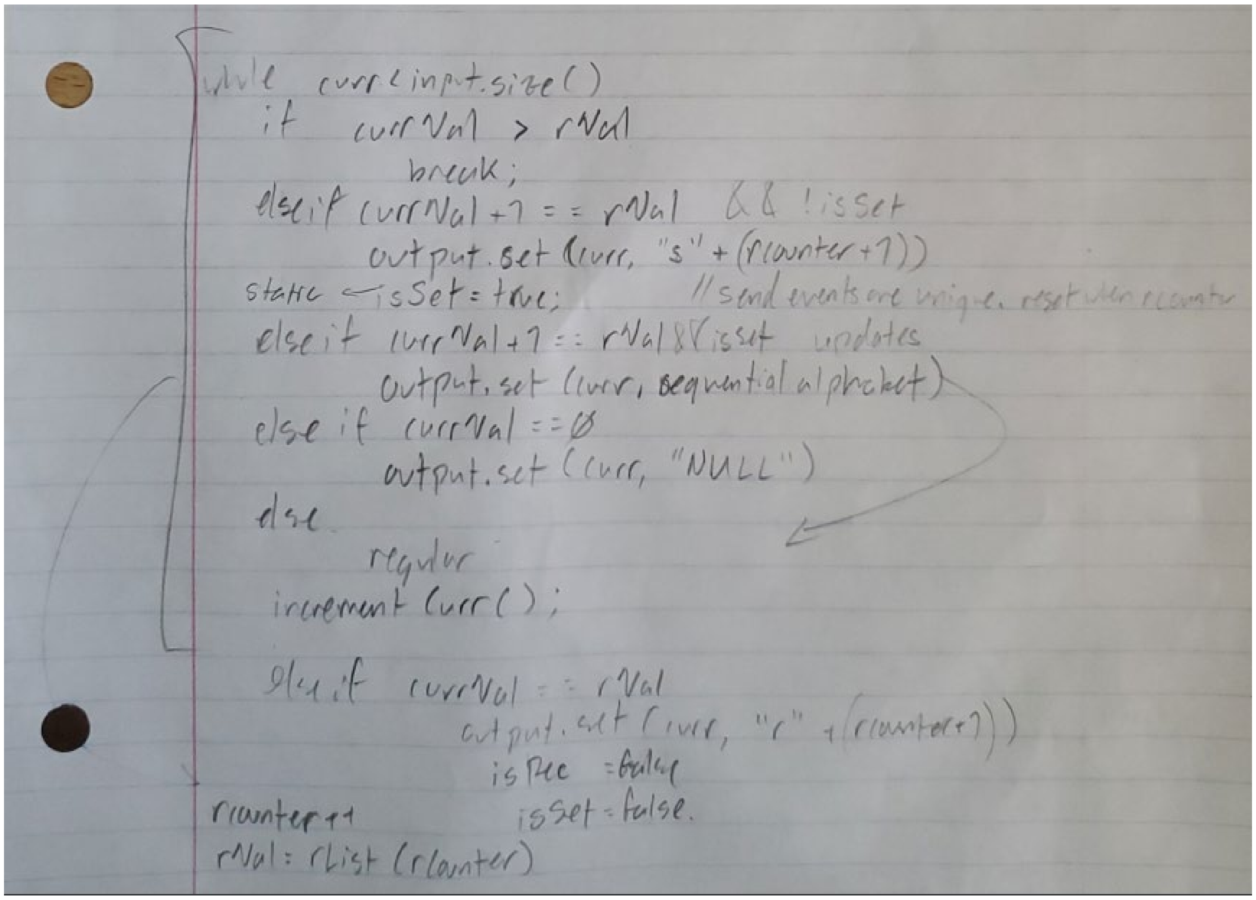
```
While...
while curr < input.length
if input.get(curr).length != 1                  r', s', Null
        if input.get(curr)[0] == 's'
            write to sList [input.get(curr), step]
             set Output (step)
              increment step
       elseif(input.get (curr)[0] == 'r')
            if   sList.length >= input.get(curr)[1]
           val = sList. get (input.get(curr)[1].toInt() -1) [1]
                if  val > step
                   step = val +1
                    set Output
                else
                    set Output(step)
                    increment step
             else     (if  rval > sval)
                 break  (do not increment curr)
        else         (if Null   or   bad input)
             set Output (0)

else
     set Output(step)
      increment step
```

More legible pseudocode.



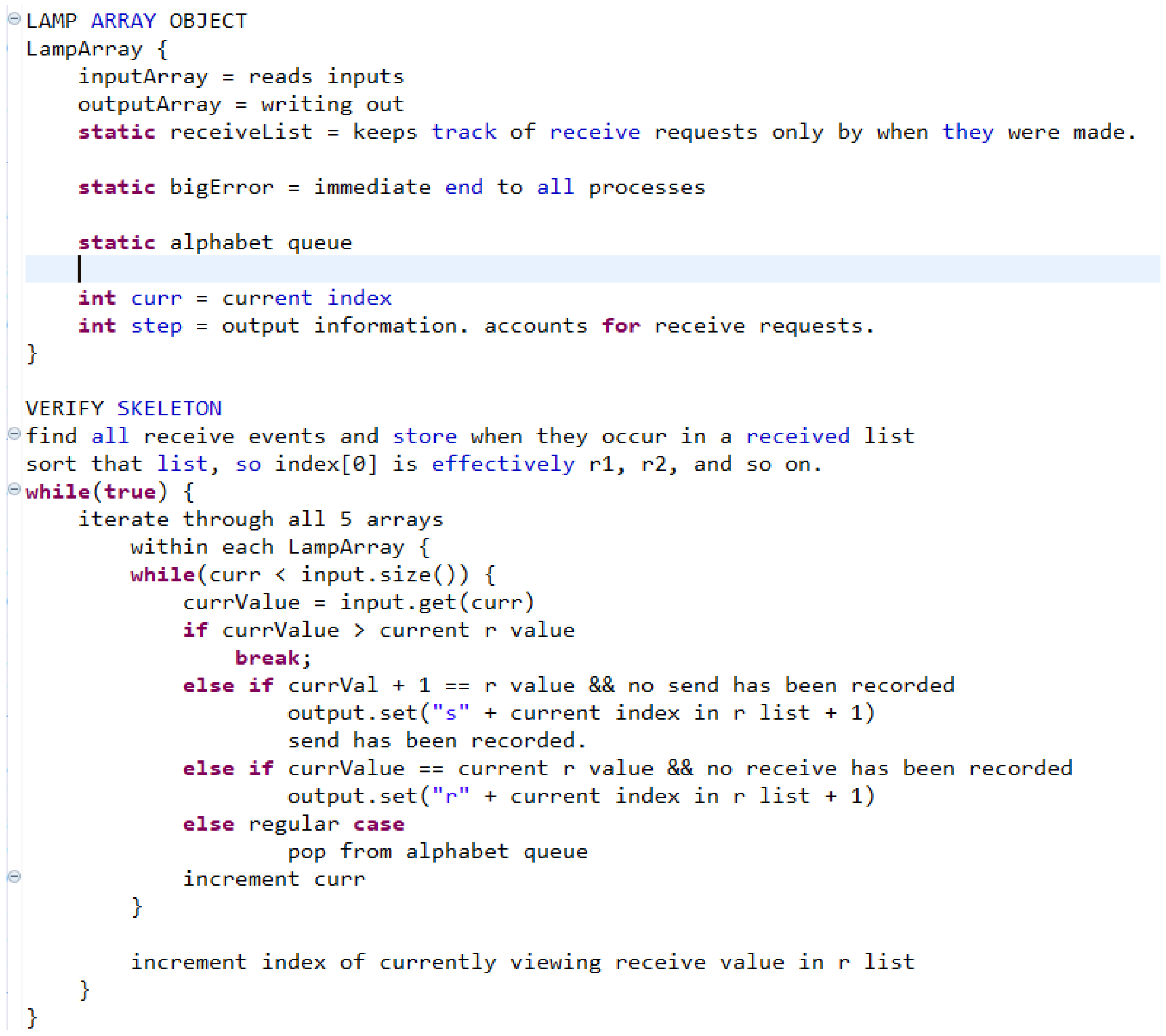
```
LAMP ARRAY OBJECT
LampArray {
    inputArray = reads inputs
    outputArray = writing out
    static sendList = keeps track of send requests and when they were made.

    static bigError = immediate end to all processes

    int curr = current index
    int step = output information. accounts for receive requests.
}

CALCULATE SKELETON
while(true) {
    loop through 5 LampArray objects until all are finished
        in each LampArray:
            while(iterator < input.length) {
                if input.get(curr).length is a special case (length != 1) {
                    if send request {
                        write to sendList the input, and when it was made.
                        setOutput(step)
                        increment counters
                    } else if receive request {
                        if request matches num with current send list {
                            val = get corresponding sendList item's time
                            if val > step {
                                step = val + 1
                                setOutput(step)
                                increment counters
                            }
                        }
                    } else if nonsense response, like NULL or mistake {
                        setOutput(0)
                    }
                } else when normal {
                    setOutput(step)
                    increment counters
                }
            }
}
```

Pseudocode used for Verify();



```
while  curr < input.size()
    if  currVal > rVal
            break;
    elseif currVal+1 == rVal  && !isSet
            output.set(curr, "s" + (rcounter+1))
static <- isSet = true;           // send events are unique, reset when rcounter
    elseif  currVal+1 == rVal && isSet   updates
            output.set(curr, sequential alphabet)
    else if  currVal == Ø
            output.set(curr, "NULL")
    else
            regular
    increment curr();

    else if  currVal == rVal
                output.set(curr, "r" + (rcounter+1))
                isRec = false
rcounter++              isSet = false.
rVal = rList(rcounter)
```

More legible pseudocode.



```
LAMP ARRAY OBJECT
LampArray {
    inputArray = reads inputs
    outputArray = writing out
    static receiveList = keeps track of receive requests only by when they were made.

    static bigError = immediate end to all processes

    static alphabet queue

    int curr = current index
    int step = output information. accounts for receive requests.
}

VERIFY SKELETON
find all receive events and store when they occur in a received list
sort that list, so index[0] is effectively r1, r2, and so on.
while(true) {
    iterate through all 5 arrays
        within each LampArray {
        while(curr < input.size()) {
            currValue = input.get(curr)
            if currValue > current r value
                break;
            else if currVal + 1 == r value && no send has been recorded
                    output.set("s" + current index in r list + 1)
                    send has been recorded.
            else if currValue == current r value && no receive has been recorded
                    output.set("r" + current index in r list + 1)
            else regular case
                    pop from alphabet queue
            increment curr
        }

        increment index of currently viewing receive value in r list
    }
}
```

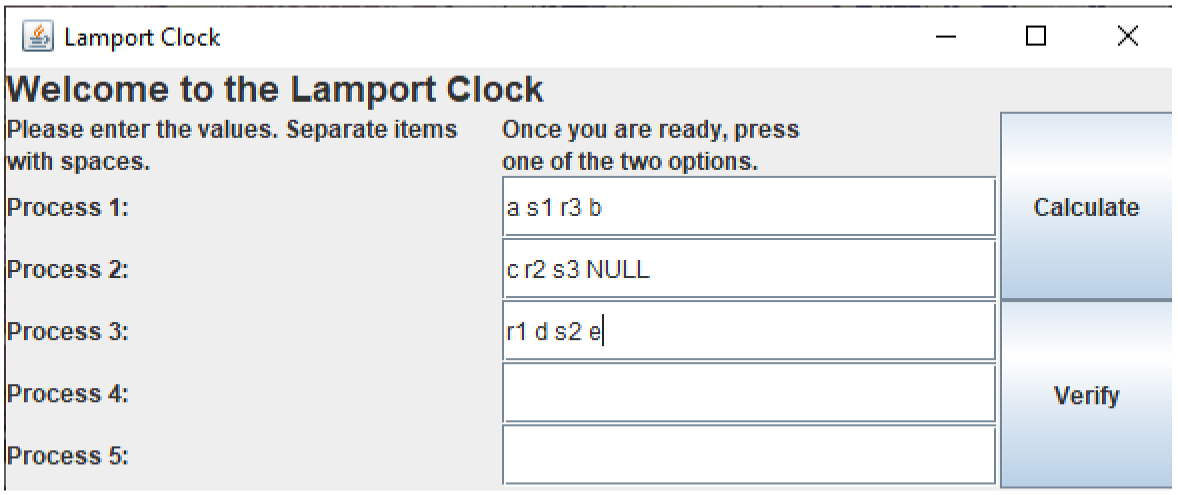
Instructions to run

Download the executable .jar file present in the Github Repository titled *MoonMichaelLamportClock.jar* located at https://github.com/CSUF-CPSC-Bein-FA21/project-1-servers-michael-moon.git

Open with JDK/JRE | Compiled in Java ver 17, compatible with Java SE-16.

Screenshots

Main Menu.



Welcome to the Lamport Clock

Please enter the values. Separate items with spaces.

Once you are ready, press one of the two options.

Process 1: a s1 r3 b

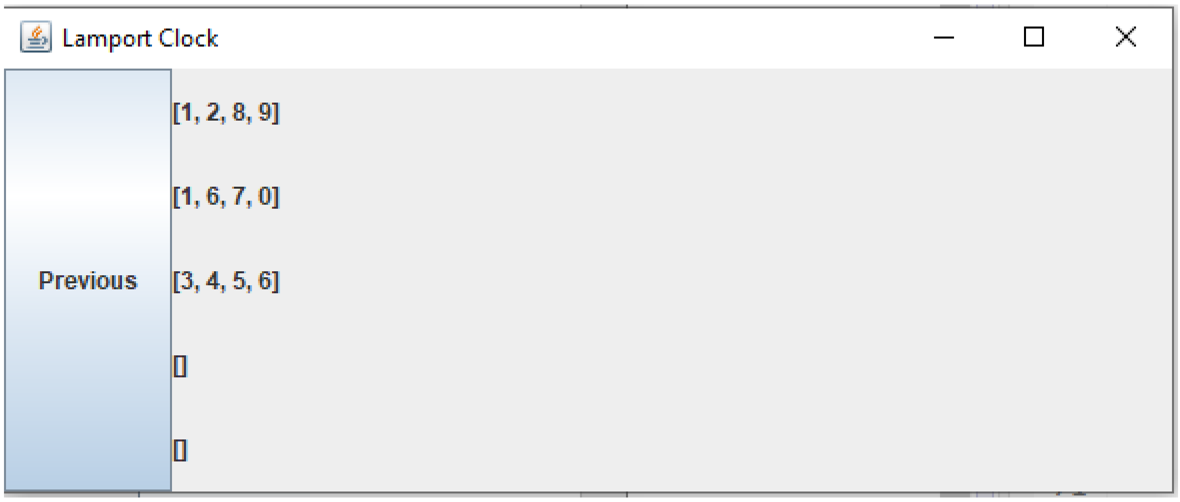
Process 2: c r2 s3 NULL

Process 3: r1 d s2 e

Process 4:

Process 5:

Upon clicking on the *Calculate* Button, the output is displayed in the same window.

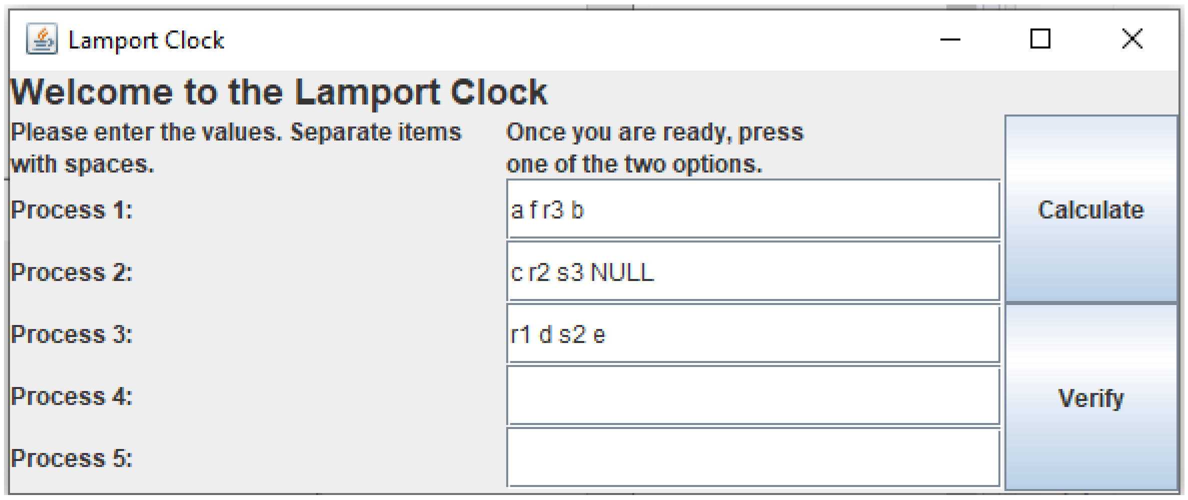


```
[1, 2, 8, 9]

[1, 6, 7, 0]

[3, 4, 5, 6]

[]

[]
```

Upon clicking on the *Previous* Button, the user is taken back to the main menu. Notice the input fields are not cleared – this is in case of an error on the user's part and allows quick editing.

The program can execute all operations in one lifetime multiple times.

The program will produce a generic error message in the event of a fail state, notice *'s1'* is replaced with *'f'*


Lamport Clock

Welcome to the Lamport Clock

Please enter the values. Separate items with spaces.

Once you are ready, press one of the two options.

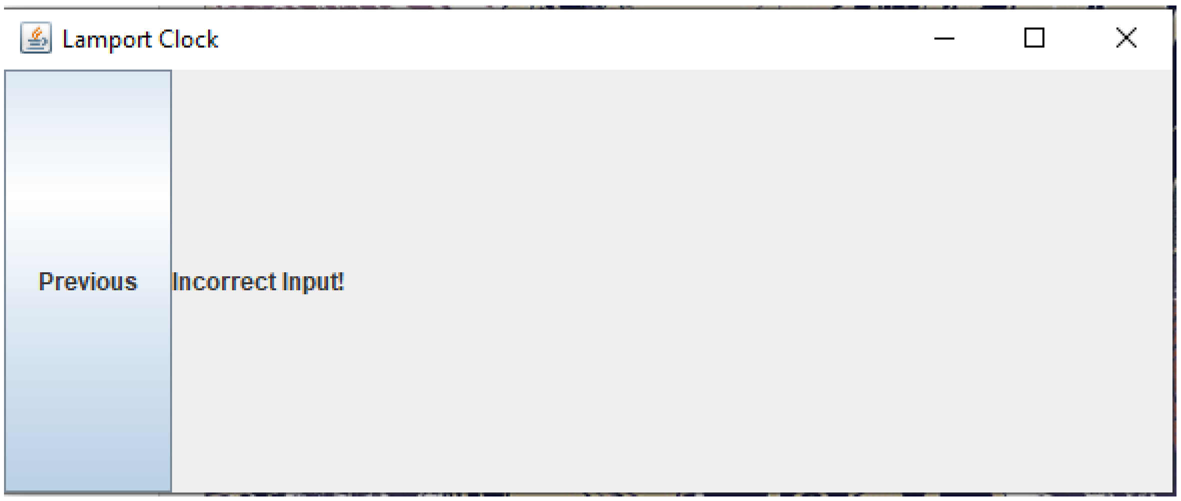
Process 1: a f r3 b

Process 2: c r2 s3 NULL

Process 3: r1 d s2 e

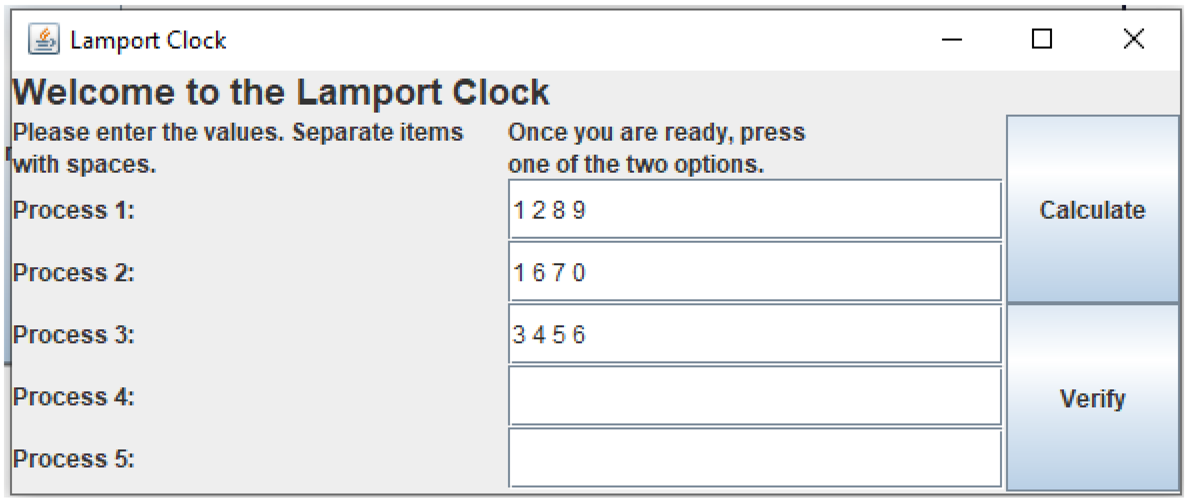
Process 4:

Process 5:


Lamport Clock

Incorrect Input!

For verify


Lamport Clock

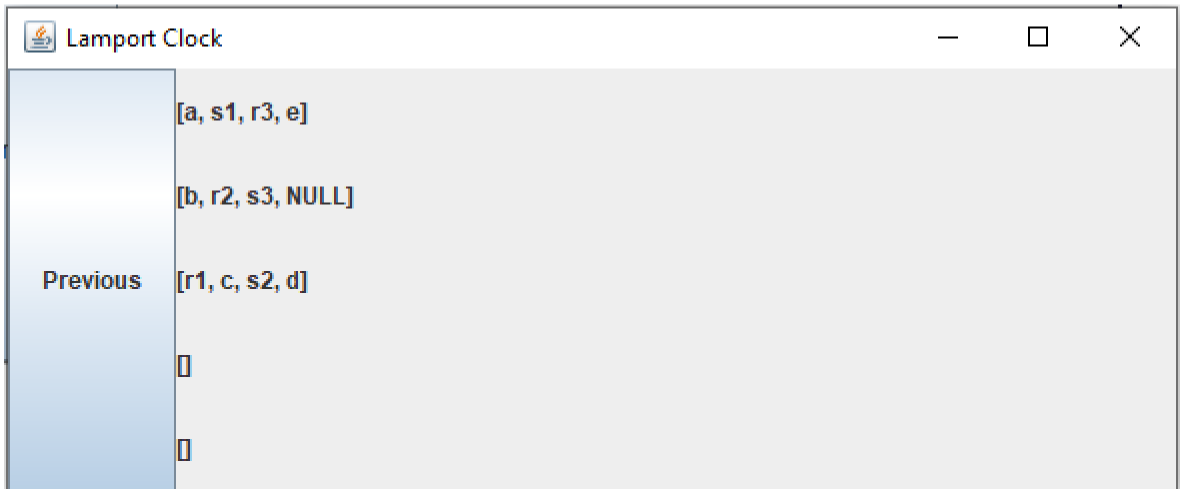
Welcome to the Lamport Clock

Please enter the values. Separate items with spaces.

Once you are ready, press one of the two options.

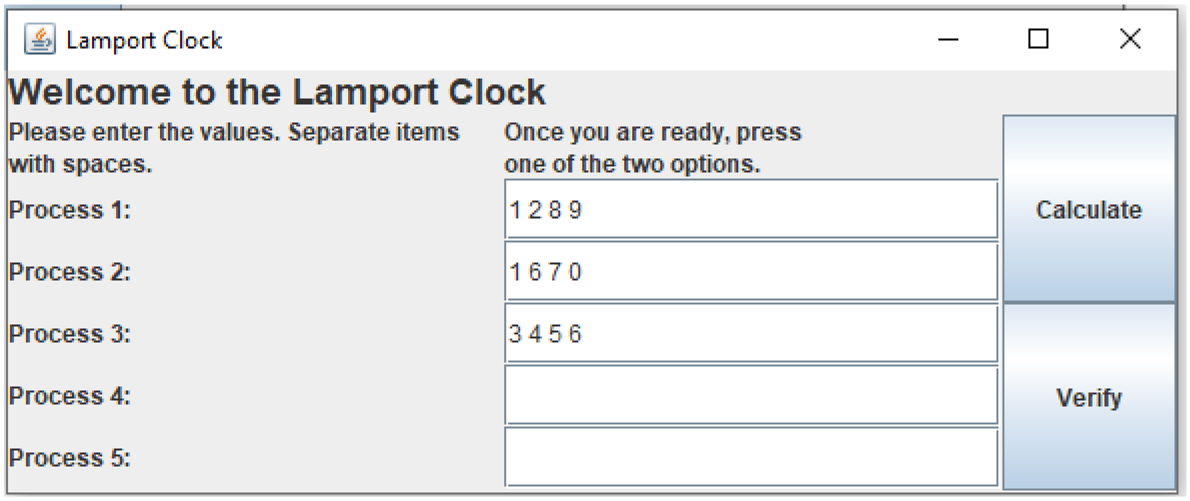
Process 1:	1 2 8 9
Process 2:	1 6 7 0
Process 3:	3 4 5 6
Process 4:	
Process 5:	

Upon clicking the *Verify* Button, users are greeted with the translated verification table. Notice the generic tasks denoted by the alphabetic characters are in line with the timing rather than filled out sequentially from array 1 to the end. Letters are filled in much the same form as the send and receive requests to hasten the program and avoid filling out send, receive and generic requests separately.


Lamport Clock

```
[a, s1, r3, e]

[b, r2, s3, NULL]

[r1, c, s2, d]

[]

[]
```

Upon clicking the *Previous* Button, users are taken back to the main menu.



Lamport Clock

Welcome to the Lamport Clock

Please enter the values. Separate items with spaces.

Once you are ready, press one of the two options.

Process 1: 1 2 8 9

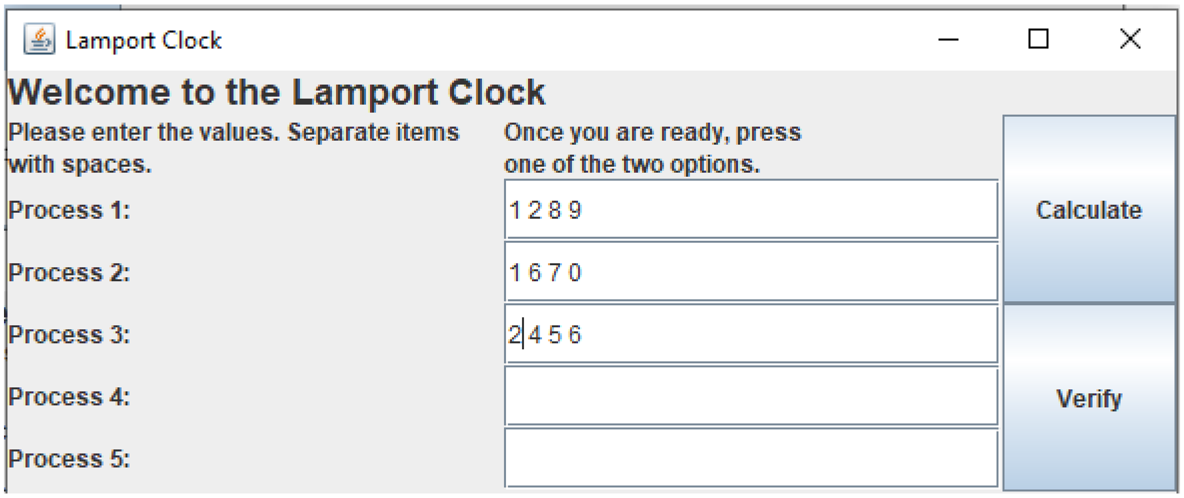
Process 2: 1 6 7 0

Process 3: 3 4 5 6

Process 4:

Process 5:

As with Calculate, erroneous inputs will be met with a generic “Incorrect!” message. Notice *input 1* from *Array 3* is a *2*, and *input 2* from *Array 3* is a *4*, making this timing sheet inaccurate.



Lamport Clock

Welcome to the Lamport Clock

Please enter the values. Separate items with spaces.

Once you are ready, press one of the two options.

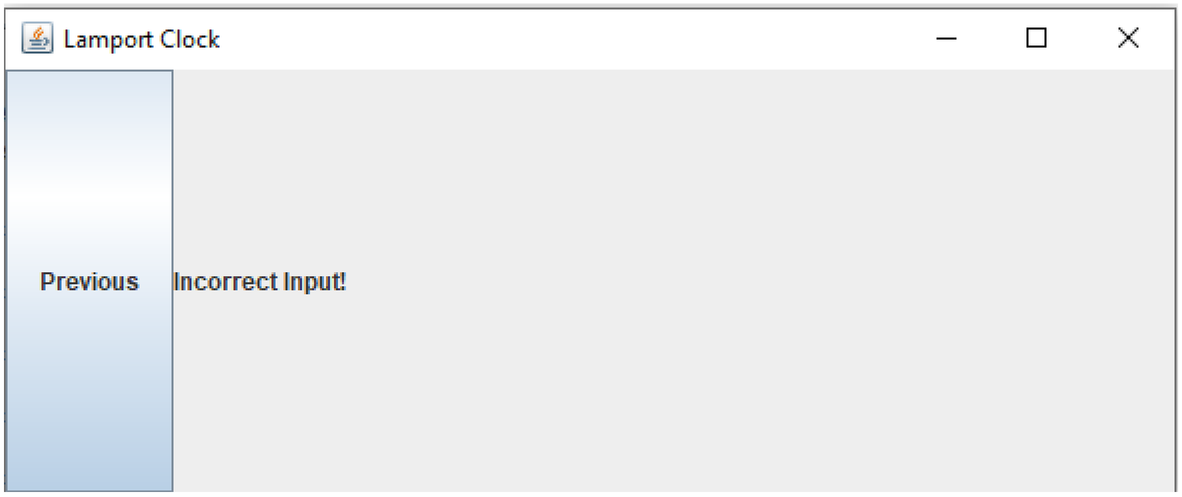
Process 1: 1 2 8 9

Process 2: 1 6 7 0

Process 3: 2 4 5 6

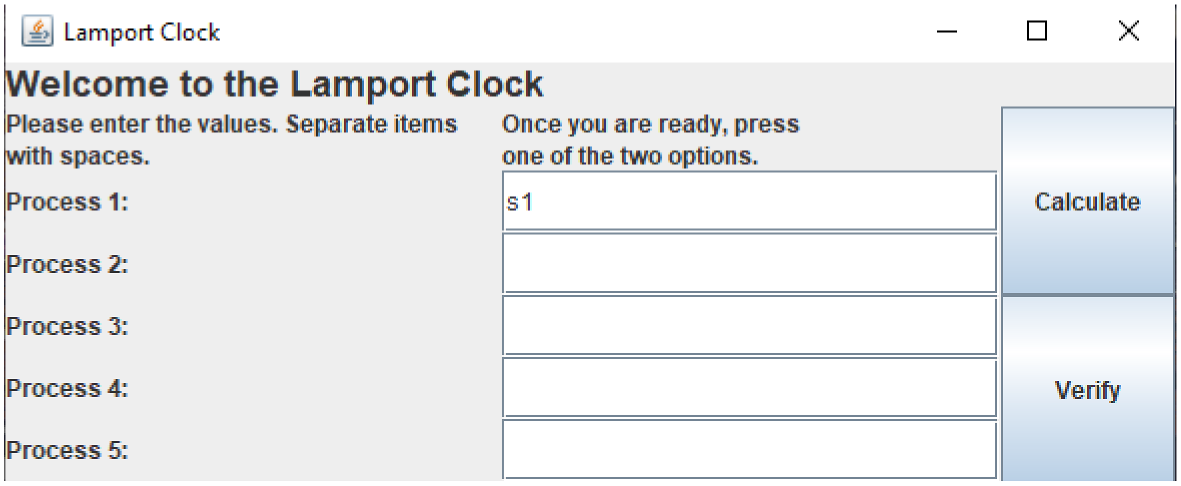
Process 4:

Process 5:



Incorrect Input!

Erroneous input applied to Verify is also met with an error message.



Welcome to the Lamport Clock

Please enter the values. Separate items with spaces.

Once you are ready, press one of the two options.

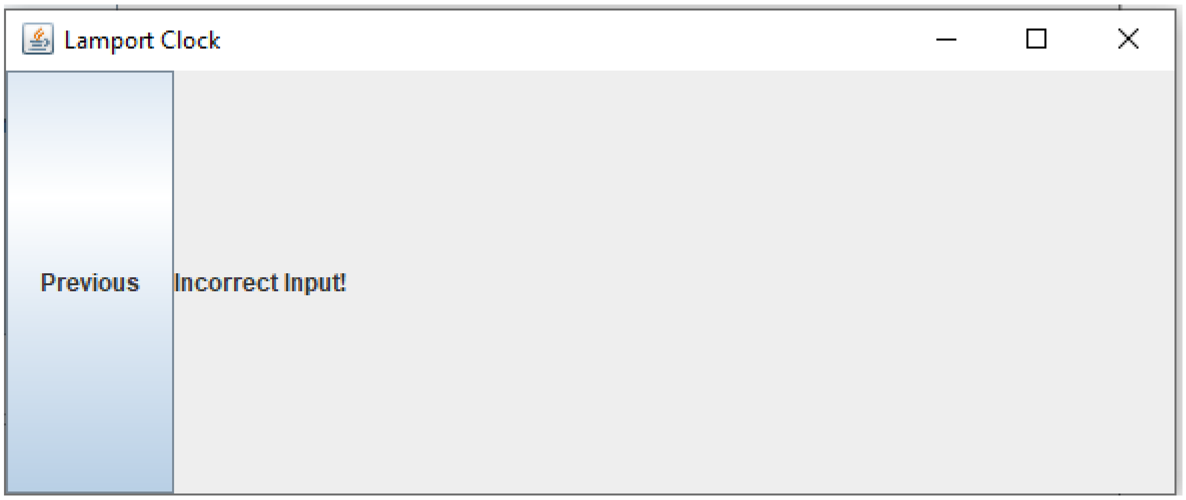
Process 1: s1

Process 2:

Process 3:

Process 4:

Process 5:



Incorrect Input!