

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

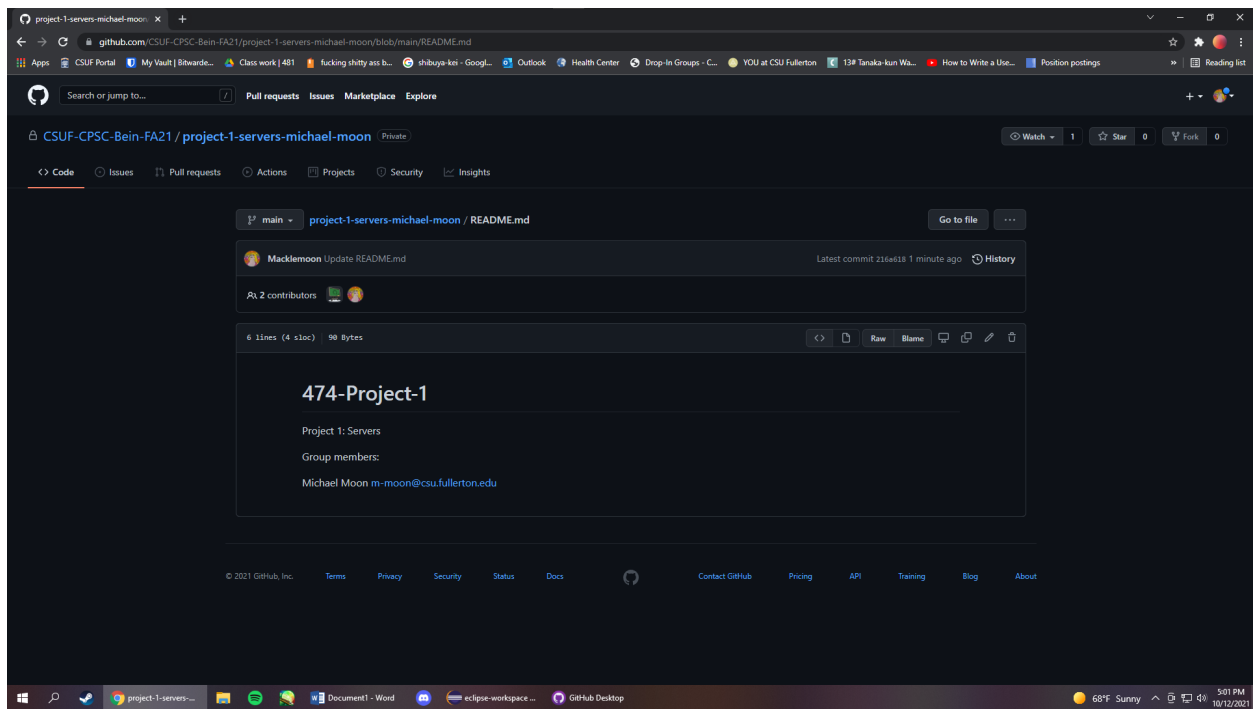
CPSC 474

Dr. Bein

15 October 2021

Project 1

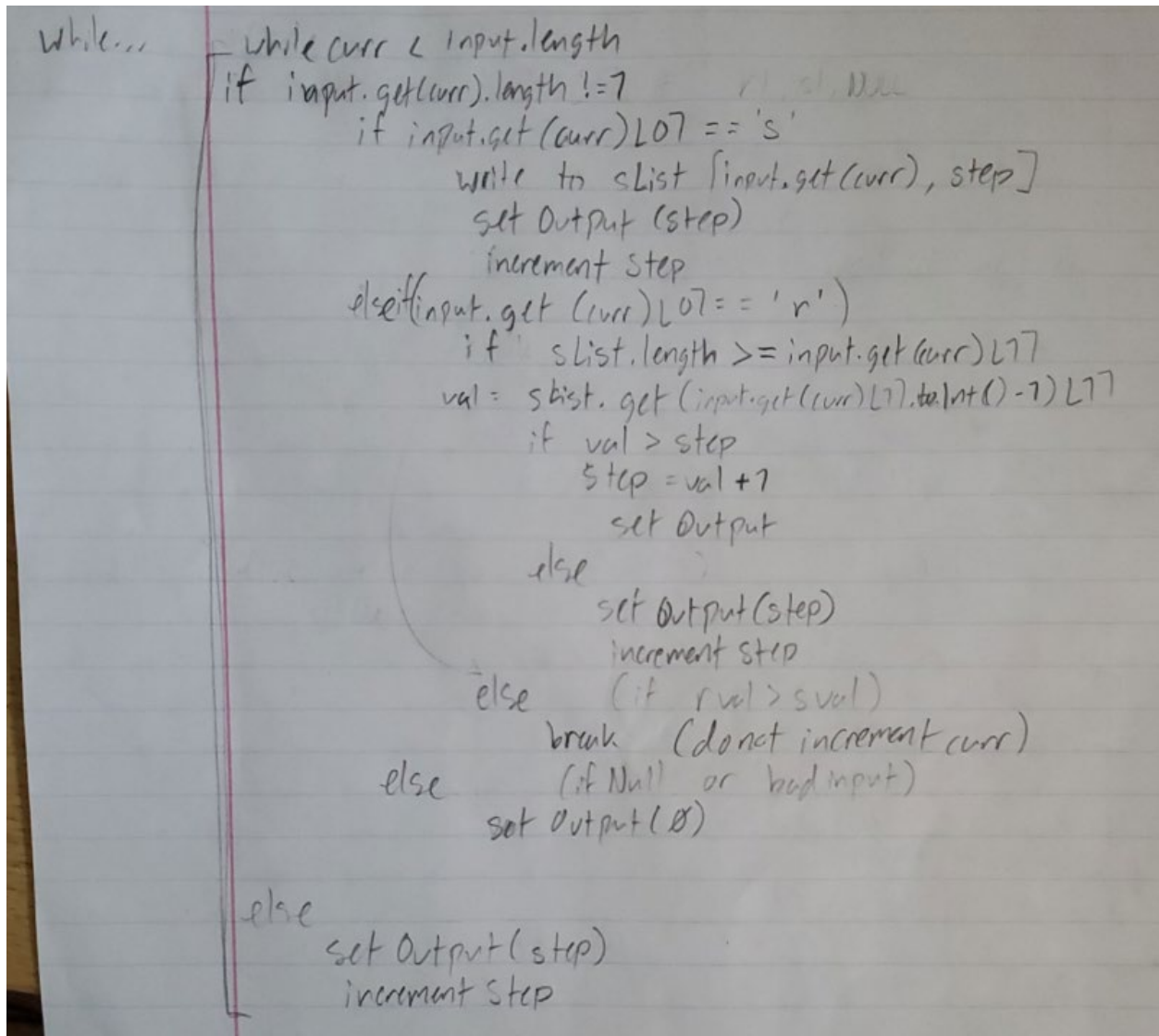
Full screenshot of team name(s), email address(es) and proof of work on Project 1.



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 != 7  
      if input.get(curr).charAt(0) == 's'  
        write to sList [input.get(curr), step]  
        set Output (step)  
        increment step  
      elseif (input.get(curr).charAt(0) == 'r')  
        if sList.length >= input.get(curr).length  
          val = sList.get(input.get(curr).length - 1).charAt(0)  
          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 currInput.size()
    if currVal > rVal
        break;
    else if currVal + 7 == rVal && !isSet
        output.set(curr, "s" + (rCounter + 7))
        state ← isSet = true; // send events are unique, reset when rCounter
    else if currVal + 7 == rVal && isSet updates
        output.set(curr, sequential alphabet)
    else if currVal == 0
        output.set(curr, "NULL")
    else
        regular
        increment(curr);

    else if currVal == rVal
        output.set(curr, "r" + (rCounter + 7))
        isRec = false;
        isSet = false;
    rCounter++
    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:	<input type="text" value="a s1 r3 b"/>	Calculate
Process 2:	<input type="text" value="c r2 s3 NULL"/>	
Process 3:	<input type="text" value="r1 d s2 e"/>	Verify
Process 4:	<input type="text"/>	
Process 5:	<input type="text"/>	

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

Previous

[1, 2, 8, 9]

[1, 6, 7, 0]

[3, 4, 5, 6]

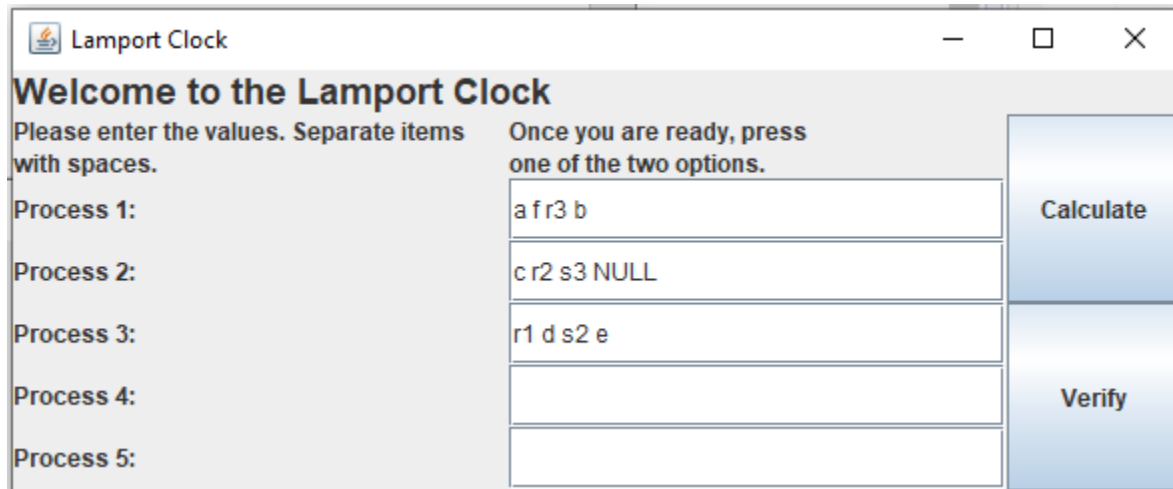
[]

[]

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.

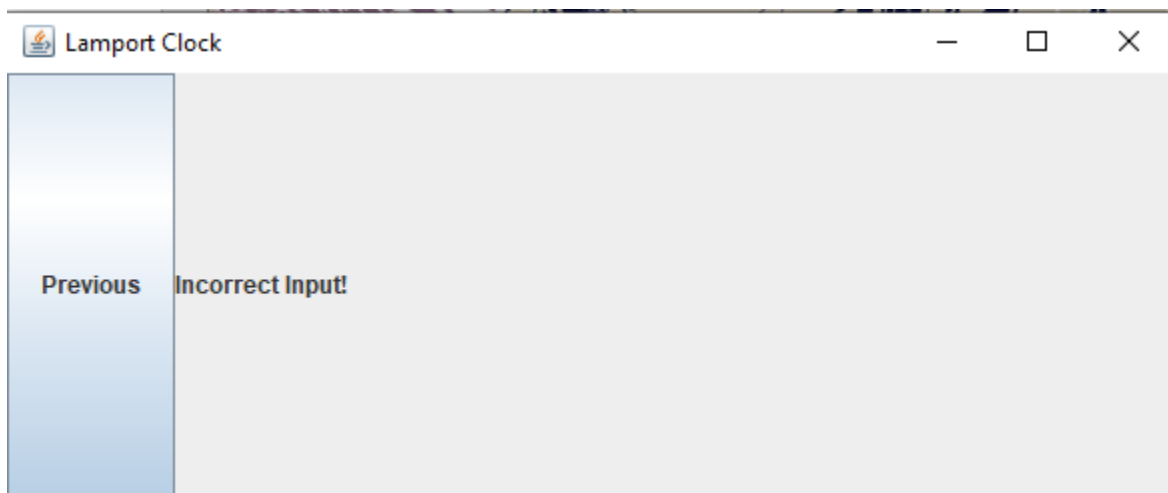
The program can execute all operations in one lifetime multiple times.

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



The screenshot shows a window titled "Lamport Clock". It has a header bar with a small icon and the title. Below the header, the text "Welcome to the Lamport Clock" is displayed. To the left of the input fields, there is a label "Please enter the values. Separate items with spaces." and a label "Once you are ready, press one of the two options." To the right of the input fields, there are two buttons: "Calculate" and "Verify". The input fields are labeled "Process 1:", "Process 2:", "Process 3:", "Process 4:", and "Process 5:". The "Calculate" button is highlighted in blue.

Process	Input
Process 1:	a f r3 b
Process 2:	c r2 s3 NULL
Process 3:	r1 d s2 e
Process 4:	
Process 5:	



The screenshot shows the same window titled "Lamport Clock". The "Previous" button is highlighted in blue. The text "Incorrect Input!" is displayed in the main area of the window.

Action	Message
Previous	Incorrect Input!

For verify

The screenshot shows a window titled "Lamport Clock". It has a header bar with a minus, maximize, and close button. Below the header, there's a title "Welcome to the Lamport Clock". The main area is divided into two columns. The left column contains instructions: "Please enter the values. Separate items with spaces." and "Once you are ready, press one of the two options." Below these are five input fields labeled "Process 1:", "Process 2:", "Process 3:", "Process 4:", and "Process 5:". The right column contains two large blue buttons: "Calculate" and "Verify". The "Calculate" button is positioned above the "Verify" button. The input fields for Process 1, 2, and 3 contain the values "1 2 8 9", "1 6 7 0", and "3 4 5 6" respectively. The input fields for Process 4 and Process 5 are empty.


Process	Values
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.

The screenshot shows the same "Lamport Clock" window. The "Previous" button is now highlighted in blue. To the right of the button, there is a list of tasks represented as arrays: "[a, s1, r3, e]", "[b, r2, s3, NULL]", "[r1, c, s2, d]", and two empty arrays "[]".

Task
[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.

Process 1:

Process 2:

Process 3:

Process 4:


Process 5:

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

Calculate

Verify

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.

Process 1:

Process 2:

Process 3:

Process 4:

Process 5:

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

Calculate

Verify



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

The screenshot shows the "Lamport Clock" application window in its main state. The title bar is at the top. Below it is a header section with the text "Welcome to the Lamport Clock" in bold. Underneath the header, there are two columns of text. The left column says "Please enter the values. Separate items with spaces." and the right column says "Once you are ready, press one of the two options." Below these instructions are five input fields, each preceded by a label: "Process 1:", "Process 2:", "Process 3:", "Process 4:", and "Process 5:". The first input field contains the text "s1". To the right of these input fields are two large blue buttons: "Calculate" (top) and "Verify" (bottom). The "Verify" button is currently highlighted with a darker blue background.

