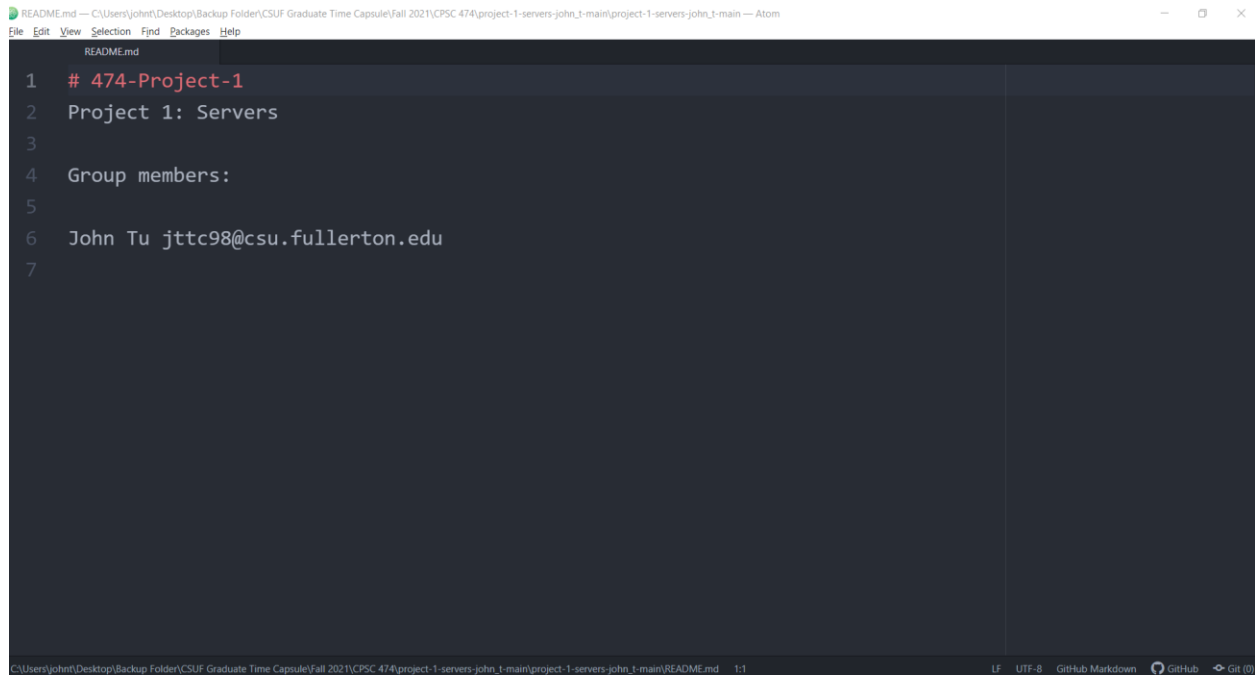


CPSC 474 Project 1 Report

John Tu (jttc98@csu.fullerton.edu)

Due October 15, 2021

Screenshot for README.md:



```
1 # 474-Project-1
2 Project 1: Servers
3
4 Group members:
5
6 John Tu jttc98@csu.fullerton.edu
7
```

Pseudocode for Calculate

void Calculate(int n, int m):

 If $((n < 1 \text{ or } n > 5) \text{ or } (m < 1 \text{ or } m > 25))$

 {

 Print "ERROR. Invalid number of processors and events. Number of processors must be at least 1 and at most 5, and there can be a maximum of 25 events for each processor." and return.

 }

 Declare a string variable called LCEvent.

 // Declare two dynamic array variables for events and values, and allocate memory.

 Declare two dynamic arrays: a string variable called inputEvents and an integer variable called outputResults. Allocate memory for those two arrays.

 for (int x = 0; x < n; x++)

```

{
    for (int y = 0; y < m; y++)
    {
        Set inputEvents[x][y] to "NULL".
    }
}

```

Print "Enter the events for each processor.".

Print "Enter NULL to skip to the next processor.".

```

for (int x = 0; x < n; x++)
{
    for (int y = 0; y < m; y++)
    {
        Have user enter event.
        If event is NULL, break and move on to next input.
        Set inputEvents[x][y] to event.
    }
}

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        Set outputResults[i][j] to o.
    }
}

```

```

for (int i = 0; i < n; i++)
{
    Print "Processor " << i << ":".
}

```

```

        for (int j = 0; j < m; j++)
        {
            Print inputEvents[i][j].
        }
        Print newline.
    }

```

Print "Now calculating the matrix...".

Declare the following integer variables: sendList[10] and numReceive.

Set numReceive to 0.

```

for (int i = 0; i < n; i++)
{
    Set sendList[i] to 0.
}

```

Count the number of receive events in input matrix.

```

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        if (inputEvents[i][j][0] == 'r')
        {
            Increment numReceive by 1.
        }
    }
}

```

```

for (int r = 0; r < numReceive; r++)

```

```

{
    for (int x = 0; x < n; x++)
    {
        Declare an integer variable called clockValue and set to 1.
        for (int y = 0; y < m; y++)
        {
            Declare a character variable called letter.
            Set letter to inputEvents[x][y][0].
            Declare an integer called sendValue and set to 0.
            if (letter == 's')
            {
                Send event found.

                // Obtain value by doing character subtraction.
                Set sendValue to inputEvents[x][y][1] - '0'.
                Store clockValue into sendList[sendValue].
                Store clockValue into outputResults[x][y].
                Increment clockValue by 1.
            }
            else if (letter == 'r')
            {
                Receive event found.
                Set sendValue to inputEvents[x][y][1] - '0'.
                if (sendList[sendValue] == 0)
                {
                    No send event found for the receive event.
                    Break and move on to the next processor.
                }
            }
            else
            {

```

If outputResults[x][y-1] > sendList[sendValue], set
clockValue to outputResults[x][y-1]. Otherwise, set clockValue to sendList[sendValue].

 Set outputResults[x][y] to clockValue+1.

 Increment clockValue by 1.

 }

}

else if (letter == 'N')

{

 Null event found, so leave it at is.

 Break out of the loop and move on to the next processor.

}

else

{

 Internal event found.

 Set outputResults[x][y] to clockValue and increment
clockValue by 1.

}

}

}

}

Print "Calculation complete. Here's the output matrix:".

for (int i = 0; i < n; i++)

{

 for (int j = 0; j < m; j++)

 {

 Print outputResults[i][j].

 }

 Print newline.

}

Print “Enter the filename for the output:”.

Have user enter a name for string variable called outputFile.

Open outputFile for write operations.

```
for (int x = 0; x < n; x++)
```

```
{
```

```
    for (int y = 0; y < m; y++)
```

```
    {
```

```
        Print outputResults[x][y] into outputFile.
```

```
    }
```

```
}
```

Close outputFile.

```
for (int i = 0; i < n; i++)
```

```
{
```

```
    Delete inputEvents[i] and outputResults[i].
```

```
}
```

Delete empty arrays inputEvents[] and outputResults[].

Return.

Pseudocode for Verify

void Verify(int n, int m):

```
    If ((n < 1 or n > 5) or (m < 1 or m > 25))
```

```
    {
```

```
        Print “ERROR. Invalid number of processors and events. Number of processors must be at least 1 and at most 5, and there can be a maximum of 25 events for each processor.” and return.
```

```
    }
```

Declare an integer variable called LCValue.

Declare two dynamic arrays: an integer variable called inputValues and a string variable called outputResults. Allocate memory for those two arrays.

```
for (int x = 0; x < n; x++)  
{  
    for (int y = 0; y < m; y++)  
    {  
        Set inputValues[x][y] to zero.  
    }  
}
```

Print "Enter the values for each processor."

Print "Enter zero to skip to the next processor."

```
for (int x = 0; x < n; x++)  
{  
    for (int y = 0; y < m; y++)  
    {  
        Have user enter value.  
        If value is NULL, break and move on to next input.  
        Set inputValue[x][y] to value.  
    }  
}  
  
for (int i = 0; i < n; i++)  
{  
    for (int j = 0; j < m; j++)  
    {  
        Set outputResults[i][j] to NULL.  
    }  
}
```

```
for (int i = 0; i < n; i++)  
{  
    Print "Processor " << i << ":".  
    for (int j = 0; j < m; j++)  
    {  
        Print inputValues[i][j].  
    }  
    Print newline.  
}
```

Declare a Boolean variable called isValidMatrix and set to true.

Print "Now verifying the matrix...".

Declare an integer array called sendList[10].

```
for (int x = 0; x < 10; x++)  
{  
    Initialize sendList[x] to 0.  
}
```

```
for (int i = 0; i < n; i++)  
{  
    if (isValidMatrix == false)  
    {  
        Break out of the loop.  
    }  
    for (int j = 0; j < m; j++)  
    {
```



```

if (j == 0)
{
    if (inputValues[i][j] == 1)
    {
        Set outputResults[i][j] to "internal".
    }
    else if (inputValues[i][j] > 1)
    {
        Set outputResults[i][j] to "r".
        Convert (j+1) to string and append to outputResults[i][j].
        Store inputValues[i][j] - 1 into sendList[j].
    }
    else if (inputValues[i][j] < 0)
    {
        // Value for input matrix cannot be negative.
        Print "ERROR. Not a valid matrix. All values in the matrix
must be positive integers." and return.

    }
}

else if (inputValues[i][j] == 0)
{
    Zero means it's a NULL event, so break out of loop.
}
else if (inputValues[i][j] < 0)
{
    // Value for input matrix cannot be negative.
    Print "ERROR. Not a valid matrix. All values in the matrix must be
positive integers." and return.
}

```

```

else
{
    if (inputValues[i][j] - inputValues[i][j - 1] == 1)
    {
        Set outputResults[i][j] to "internal".
    }
    else if (inputValues[i][j] - inputValues[i][j-1] > 1)
    {
        Set outputResults[i][j] to "r".
        Convert (j+1) to string and append to outputResults[i][j].
        Store inputValues[i][j] - 1 into sendList[j].
    }
    else if (inputValues[i][j] - inputValues[i][j-1] < 0)
    {
        // All values in each processor must be increasing order.
        Set isValidMatrix to false and break.
    }
}
}

}

if (isValidMatrix == false)
{
    Print "ERROR. Not a valid matrix. All values in each processor must be in
increasing order." and return.
}

```

Declare an integer variable called numSendEvents and set to 0.

Declare a Boolean array called isMapped[10].

```
for (int x = 0; x < 10; x++)
```

```
{
```

```
    if (sendList[x] > 0)
```

```
    {
```

```
        Increment numSendEvents by 1.
```

```
    }
```

```
}
```

```
for (int x = 0; x < 10; x++)
```

```
{
```

```
    Set isMapped[x] to false.
```

```
}
```

```
for (int x = 0; x < numSendEvents; x++)
```

```
{
```

```
    Print "Value for send event " << x+1 << ": " << sendList[x].
```

```
}
```

```
for (int s = 0; s < numSendEvents; s++)
```

```
{
```

```
    for (int x = 0; x < n; x++)
```

```
    {
```

```
        for (int y = 0; y < m; y++)
```

```
        {
```

```
            if (sendList[s] == inputValues[x][y] and isMapped[s] != true)
```

```
            {
```

```
                Set outputResults[x][y] to "s".
```

```
                Convert (s+1) to string and append to outputResults[x][y].
```

Set isMapped[s] to true.

}

}

}

}

for (int i = 0; i < numSendEvents; i++)

{

if (isMapped[i] != true)

{

Set isValidMatrix to false.

Break out of loop.

}

}

if (isValidMatrix == false)

{

Print "ERROR. Not a valid matrix. There must be a corresponding send event for each receive event."

}

else

{

Declare a character variable called internalCharacter and set to 'a'.

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

if (outputResults[i][j] == "internal")

{

Set outputResults[i][j] to internalCharacter.

Increment internalCharacter by 1.

}

}

}

Print "Verification complete. Here's the output matrix:".

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

Print outputResults[i][j].

}

Print newline.

}

Print "Enter the filename for the output:".

Have user enter a name for string variable called outputFile.

Open outputFile for write operations.

for (int x = 0; x < n; x++)

{

for (int y = 0; y < m; y++)

{

Print outputResults[x][y] into outputFile.

}

}

Close outputFile.

}

for (int i = 0; i < n; i++)

```
{  
    Delete inputValues[i] and outputResults[i].  
}
```

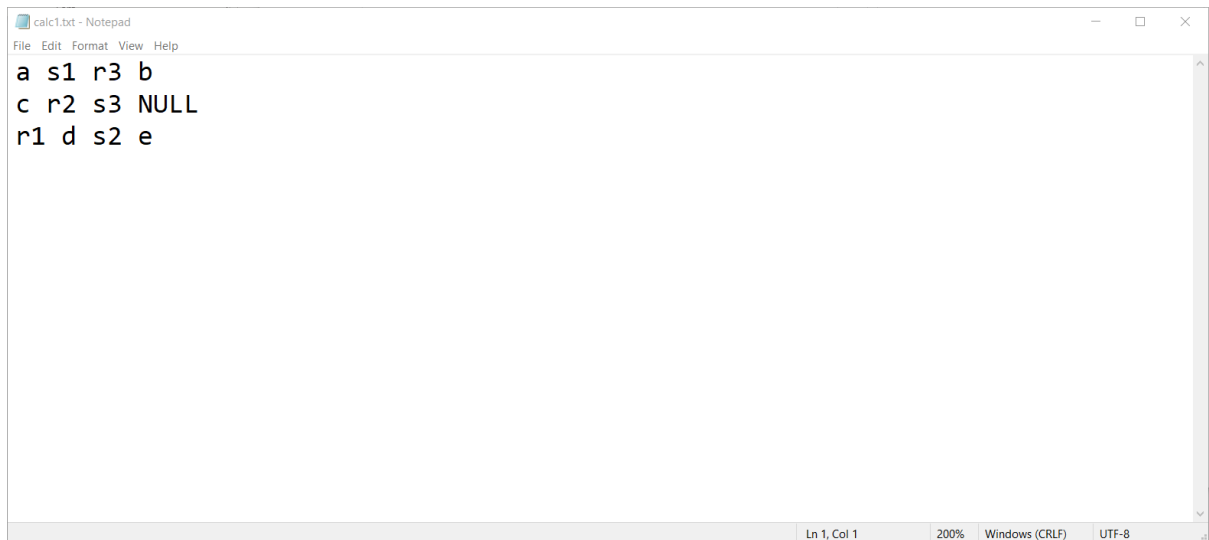
Delete empty arrays inputValues[] and outputResults[].

Return.

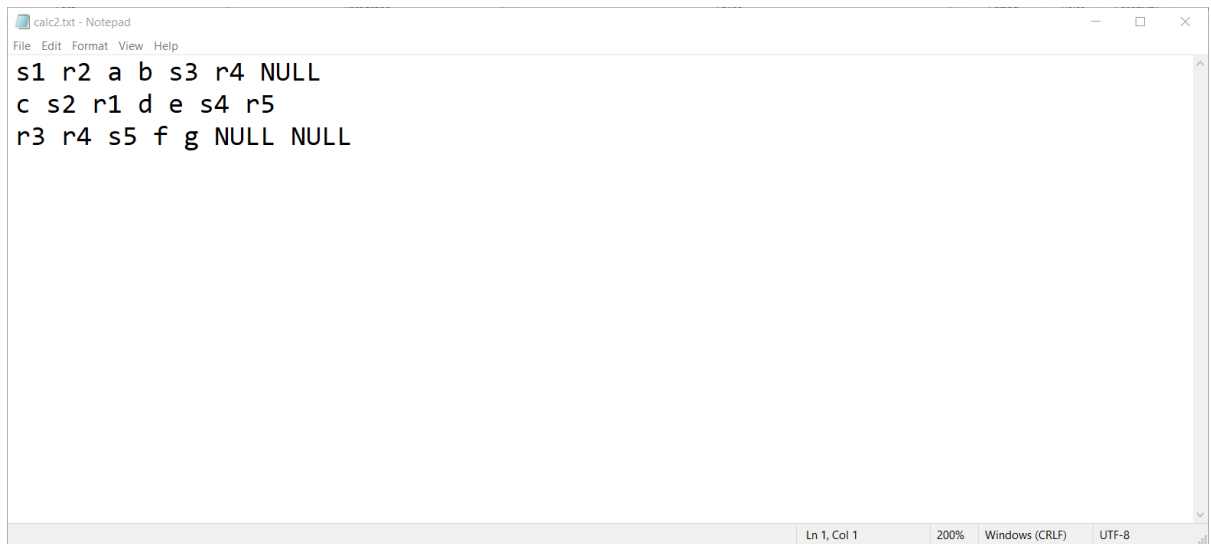
How to run the program on Windows:

Download the whole repository from GitHub and add the source code file called lamport_clock.cpp to the Visual Studio source files folder inside the Solutions Explorer window. After that, click on Start without Debugging button in the Debug drop-down menu to run the code (or press Ctrl+F5 to run). Also, open the following text files for Calculate and Verify:

- calc1.txt

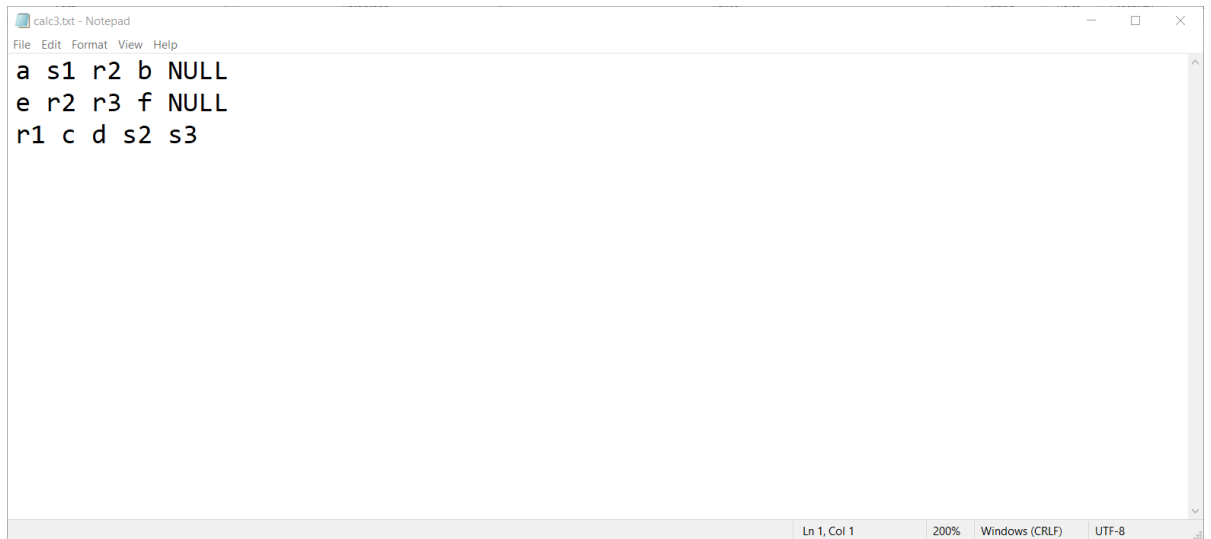


- **calc2.txt**



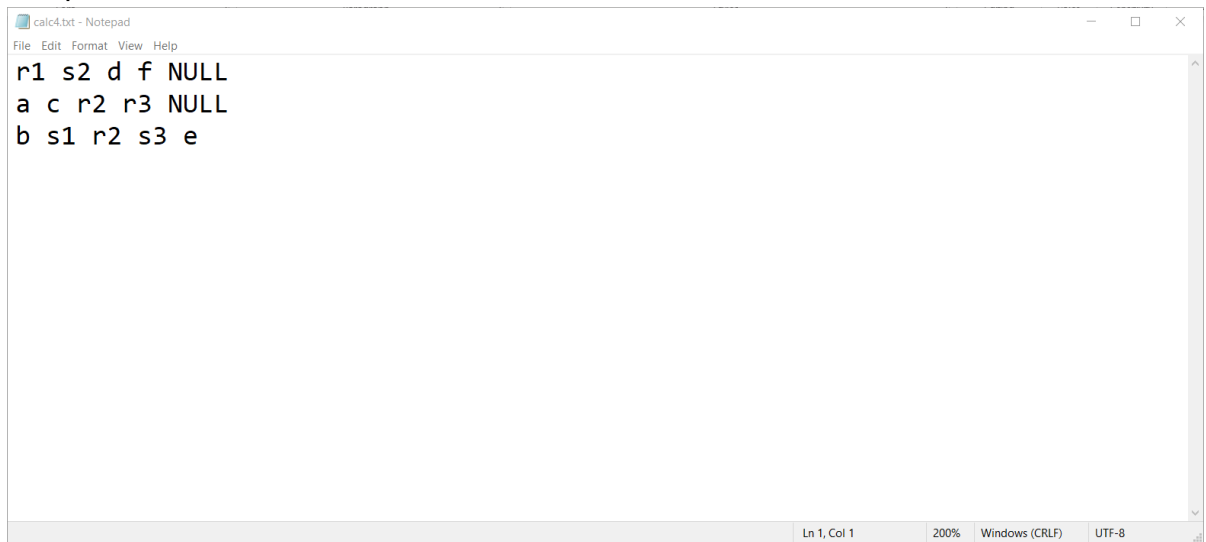
```
calc2.txt - Notepad
File Edit Format View Help
s1 r2 a b s3 r4 NULL
c s2 r1 d e s4 r5
r3 r4 s5 f g NULL NULL
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

- **calc3.txt**



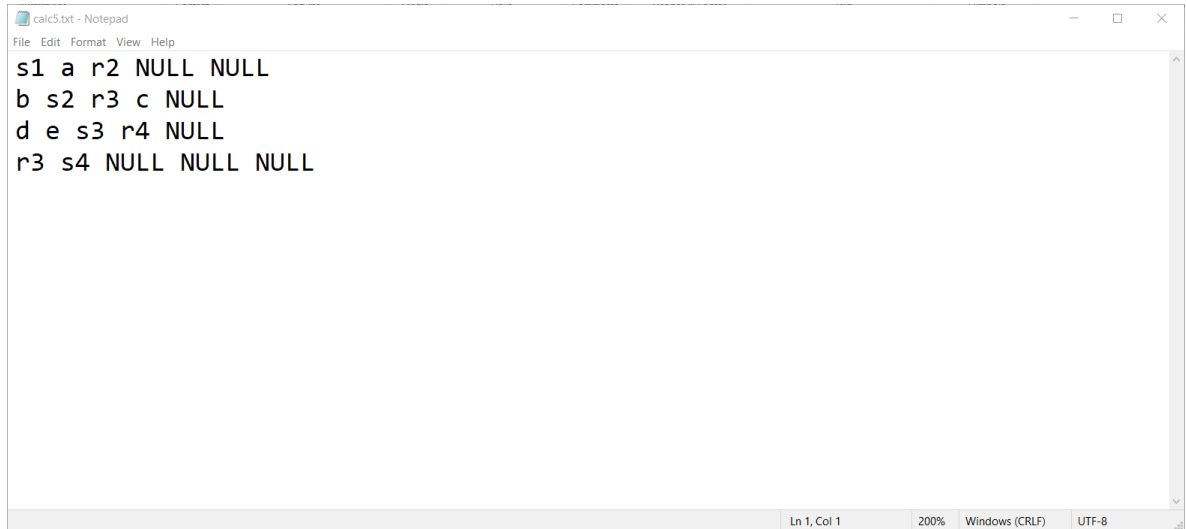
```
calc3.txt - Notepad
File Edit Format View Help
a s1 r2 b NULL
e r2 r3 f NULL
r1 c d s2 s3
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

- **calc4.txt**



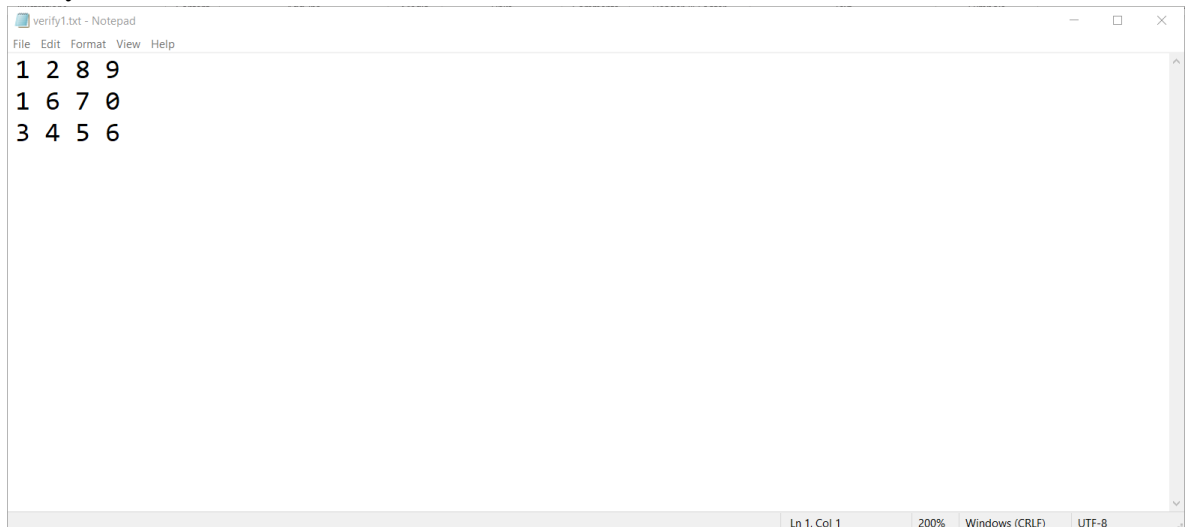
```
calc4.txt - Notepad
File Edit Format View Help
r1 s2 d f NULL
a c r2 r3 NULL
b s1 r2 s3 e
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

- **calc5.txt**



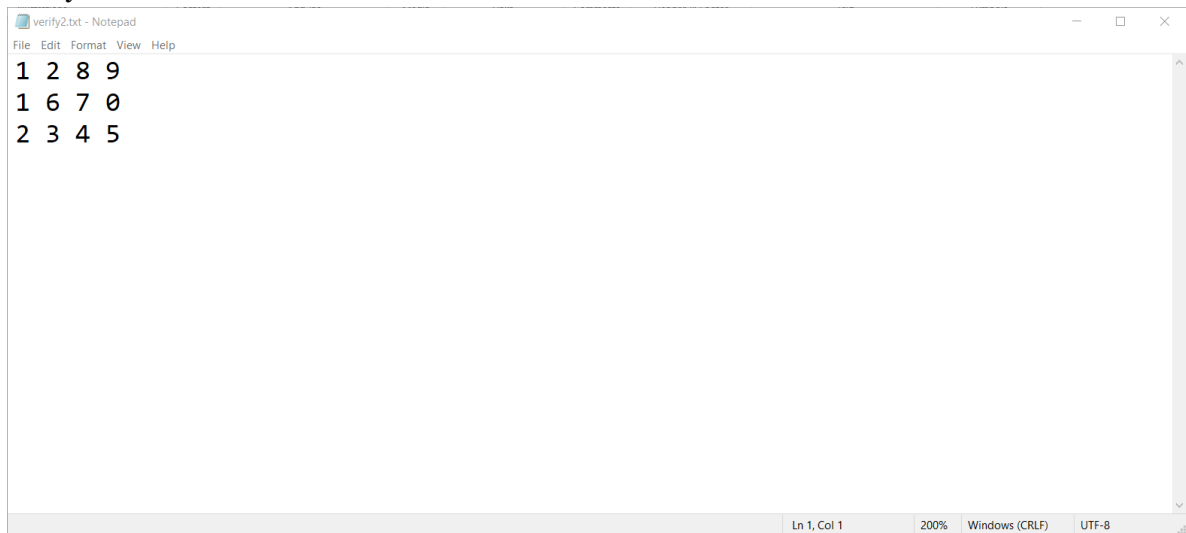
```
calc5.txt - Notepad
File Edit Format View Help
s1 a r2 NULL NULL
b s2 r3 c NULL
d e s3 r4 NULL
r3 s4 NULL NULL NULL
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

- **verify1.txt**



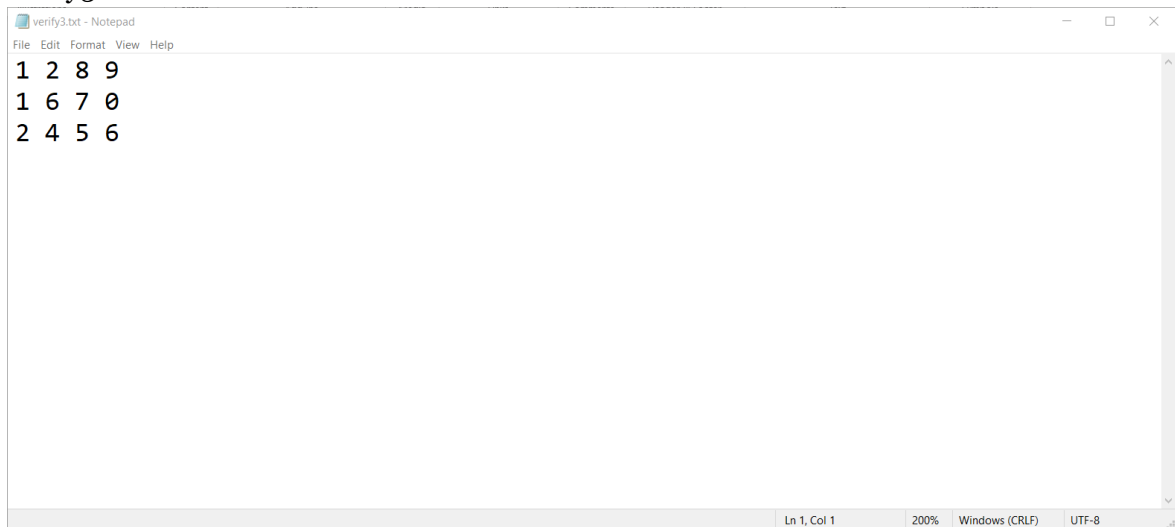
```
verify1.txt - Notepad
File Edit Format View Help
1 2 8 9
1 6 7 0
3 4 5 6
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```


- **verify2.txt**



```
1 2 8 9
1 6 7 0
2 3 4 5
```

- **verify3.txt**



```
1 2 8 9
1 6 7 0
2 4 5 6
```

Each line in the text file represents processors, and each element in the processor represents an event for Calculate input or the value of the event for the Verify input.

Count the number of lines in the text file, and that will be n number of processors. For each line, count the number of elements separated by whitespace, and that will be m number of events per processor. Enter those two numbers for the Calculate and Verify prompt. After that, for each current processor, type in each event for Calculate or value of event for Verify and press Enter to move on to the next event. To skip to the next processor, type NULL for Calculate or zero for Verify and press Enter.

Screenshots for Test Cases

Calculate

calc1.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Enter the number of processors (max 5 processors): 3
Enter the number of events (max 25 events): 4
Enter the events for each processor.
Enter NULL to skip to the next processor.
Processor 0:
Event 1: a
Event 2: s1
Event 3: r3
Event 4: b
Processor 1:
Event 1: c
Event 2: r2
Event 3: s3
Event 4: NULL
Processor 2:
Event 1: r1
Event 2: d
Event 3: s2
Event 4: e
Processor 0:      a      s1      r3      b
Processor 1:      c      r2      s3      NULL
Processor 2:      r1      d      s2      e
Now calculating the matrix...
Calculation complete. Here's the output matrix:
  1   2   8   9
  1   6   7   0
  3   4   5   6
Enter the file name for the results array: results_calc1.txt
Output written into file.
```

Output for calc1.txt:

```
results_calc1.txt - Notepad
File Edit Format View Help
  1   2   8   9
  1   6   7   0
  3   4   5   6
Ln 4, Col 1 200% Windows (CRLF) UTF-8
```

calc2.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Event 3: a
Event 4: b
Event 5: s3
Event 6: r4
Event 7: NULL
Processor 1:
Event 1: c
Event 2: s2
Event 3: r1
Event 4: d
Event 5: e
Event 6: s4
Event 7: r5
Processor 2:
Event 1: r3
Event 2: r4
Event 3: s5
Event 4: f
Event 5: g
Event 6: NULL
Processor 0:  s1  r2  a  b  s3  r4  NULL
Processor 1:  c  s2  r1  d  e  s4  r5
Processor 2:  r3  r4  s5  f  g  NULL  NULL
Now calculating the matrix...
Calculation complete. Here's the output matrix:
  1  3  4  5  6  7  0
  1  2  3  4  5  6  10
  7  8  9  10 11 0  0
Enter the file name for the results array: results_calc2.txt
Output written into file.
```

Output for calc2.txt:

```
results_calc2.txt - Notepad
File Edit Format View Help
  1  3  4  5  6  7  0
  1  2  3  4  5  6  10
  7  8  9  10 11 0  0
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

calc3.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Enter the events for each processor.
Enter NULL to skip to the next processor.
Processor 0:
Event 1: a
Event 2: s1
Event 3: r2
Event 4: b
Event 5: NULL
Processor 1:
Event 1: e
Event 2: r2
Event 3: r3
Event 4: f
Event 5: NULL
Processor 2:
Event 1: r1
Event 2: c
Event 3: d
Event 4: s2
Event 5: s3
Processor 0:      a      s1      r2      b      NULL
Processor 1:      e      r2      r3      f      NULL
Processor 2:      r1      c      d      s2      s3
Now calculating the matrix...
Calculation complete. Here's the output matrix:
  1   2   7   8   0
  1   7   8   9   0
  3   4   5   6   7
Enter the file name for the results array: results_calc3.txt
Output written into file.
```

Output for calc3.txt:

```
results_calc3.txt - Notepad
File Edit Format View Help
  1   2   7   8   0
  1   7   8   9   0
  3   4   5   6   7
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

calc4.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Enter the events for each processor.
Enter NULL to skip to the next processor.
Processor 0:
Event 1: r1
Event 2: s2
Event 3: d
Event 4: f
Event 5: NULL
Processor 1:
Event 1: a
Event 2: c
Event 3: r2
Event 4: r3
Event 5: NULL
Processor 2:
Event 1: b
Event 2: s1
Event 3: r2
Event 4: s3
Event 5: e
Processor 0:  r1  s2  d  f  NULL
Processor 1:  a   c  r2  r3  NULL
Processor 2:  b  s1  r2  s3   e
Now calculating the matrix...
Calculation complete. Here's the output matrix:
  3   4   5   6   0
  1   2   5   7   0
  1   2   5   6   7
Enter the file name for the results array: results_calc4.txt
Output written into file.
```

Output for calc4.txt:

```
results_calc4.txt - Notepad
File Edit Format View Help
  3   4   5   6   0
  1   2   5   7   0
  1   2   5   6   7
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

calc5.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Event 3: r2
Event 4: NULL
Processor 1:
Event 1: b
Event 2: s2
Event 3: r3
Event 4: c
Event 5: NULL
Processor 2:
Event 1: d
Event 2: e
Event 3: s3
Event 4: r4
Event 5: NULL
Processor 3:
Event 1: r3
Event 2: s4
Event 3: NULL
Processor 0:  s1  a  r2  NULL  NULL
Processor 1:  b  s2  r3  c  NULL
Processor 2:  d  e  s3  r4  NULL
Processor 3:  r3  s4  NULL  NULL  NULL
Now calculating the matrix...
Calculation complete. Here's the output matrix:
  1  2  3  0  0
  1  2  4  5  0
  1  2  3  6  0
  4  5  0  0  0
Enter the file name for the results array: results_calc5.txt
Output written into file.
```

Output for calc5.txt:

```
results_calc5.txt - Notepad
File Edit Format View Help
  1  2  3  0  0
  1  2  4  5  0
  1  2  3  6  0
  4  5  0  0  0
Ln 1, Col 1 200% Windows (CRLF) UTF-8
```

Verify

verify1.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Enter the values for each processor.
Enter zero to skip to the next processor.
Processor 0:
Value 1: 1
Value 2: 2
Value 3: 8
Value 4: 9
Processor 1:
Value 1: 1
Value 2: 6
Value 3: 7
Value 4: 0
Processor 2:
Value 1: 3
Value 2: 4
Value 3: 5
Value 4: 6
Processor 0:      1      2      8      9
Processor 1:      1      6      7      0
Processor 2:      3      4      5      6
Now verifying the matrix...
Value for send event 1: 2
Value for send event 2: 5
Value for send event 3: 7
Verification complete. Here's the output matrix:
  a   s1   r3   b
  c   r2   s3  NULL
 r1   d   s2   e
Enter the file name for the results array: results_verify1.txt
Output written into file.
```

Output for verify1.txt:

```
results_verify1.txt - Notepad
File Edit Format View Help
  a   s1   r3   b
  c   r2   s3  NULL
 r1   d   s2   e
Ln 1, Col 1  200%  Windows (CRLF)  UTF-8
```

verify2.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Enter the values for each processor.
Enter zero to skip to the next processor.
Processor 0:
Value 1: 1
Value 2: 2
Value 3: 8
Value 4: 9
Processor 1:
Value 1: 1
Value 2: 6
Value 3: 7
Value 4: 0
Processor 2:
Value 1: 2
Value 2: 3
Value 3: 4
Value 4: 5
Processor 0:      1      2      8      9
Processor 1:      1      6      7      0
Processor 2:      2      3      4      5
Now verifying the matrix...
Value for send event 1: 1
Value for send event 2: 5
Value for send event 3: 7
Verification complete. Here's the output matrix:
s1   a   r3   b
c    r2   s3  NULL
r1   d    e   s2
Enter the file name for the results array: results_verify2.txt
Output written into file.
```

Output for verify2.txt:

```
results_verify2.txt - Notepad
File Edit Format View Help
s1   a   r3   b
c    r2   s3  NULL
r1   d    e   s2
Ln 1, Col 1  200%  Windows (CRLF)  UTF-8
```


verify3.txt:

```
C:\Users\johnt\source\repos\ParallelComp\Debug\ParallelComp.exe
Enter the number of processors (max 5 processors): 3
Enter the number of events (max 25 events): 4
Enter the values for each processor.
Enter zero to skip to the next processor.
Processor 0:
Value 1: 1
Value 2: 2
Value 3: 8
Value 4: 9
Processor 1:
Value 1: 1
Value 2: 6
Value 3: 7
Value 4: 0
Processor 2:
Value 1: 2
Value 2: 4
Value 3: 5
Value 4: 6
Processor 0:    1    2    8    9
Processor 1:    1    6    7    0
Processor 2:    2    4    5    6
Now verifying the matrix...
Value for send event 1: 1
Value for send event 2: 3
Value for send event 3: 7
ERROR. Not a valid matrix.
There must be a corresponding send event for each receive event.

Please choose the following options:
```