CSE3038 Project 1
Report

Group members:

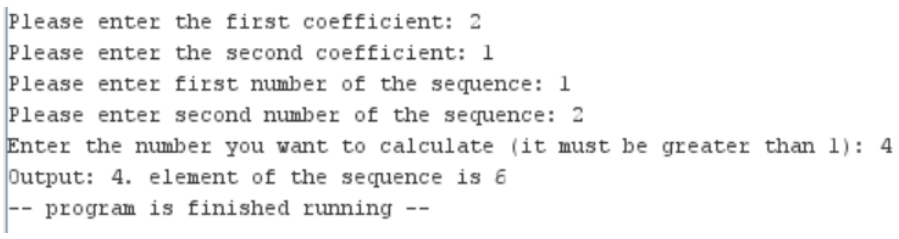
1. Mustafa Emir Uyar 150120007
2. Muhammed Hayta 150121068
3. Eren Duyuk 150120509

Question 1

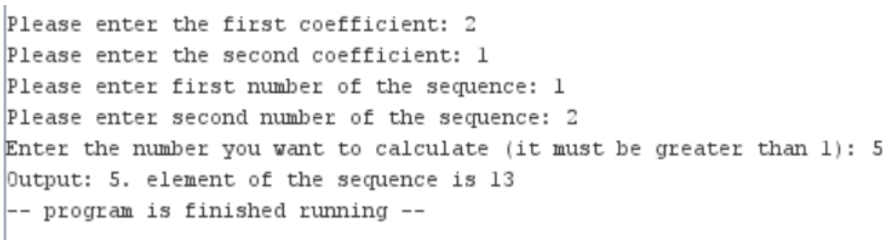
In the 1st question, we find a sequence of numbers using this function:
f(x) = a* f(x-1) + b * f(x-2) – 2
with given a,b,x0 and x1 values. The objective is to find the nth element in this sequence. the basic flow of the mips code is:

1. Take input from the user
2. recursively call the calculate sequence function
3. display the nth element

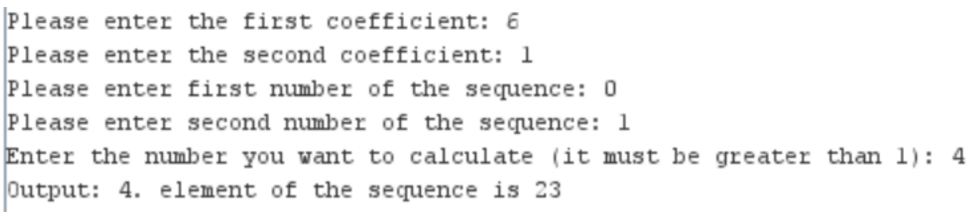
The code starts with prompting the user for the a,b,x0,x1 and n values. If the n value is less than or equal to 1, the program prompts the user again. After that it calls the recursive "calculate_sequence" function with a0 containing the variable n. The other variables are accessed from registers $s0,$s1,$s2,$s3 which contain a,b,x0,x1 respectively. after this recursive function is done it returns the nth element in the register $v0. the program then prints the nth element and terminates itself.



```
Please enter the first coefficient: 2
Please enter the second coefficient: 1
Please enter first number of the sequence: 1
Please enter second number of the sequence: 2
Enter the number you want to calculate (it must be greater than 1): 4
Output: 4. element of the sequence is 6
-- program is finished running --
```



```
Please enter the first coefficient: 2
Please enter the second coefficient: 1
Please enter first number of the sequence: 1
Please enter second number of the sequence: 2
Enter the number you want to calculate (it must be greater than 1): 5
Output: 5. element of the sequence is 13
-- program is finished running --
```



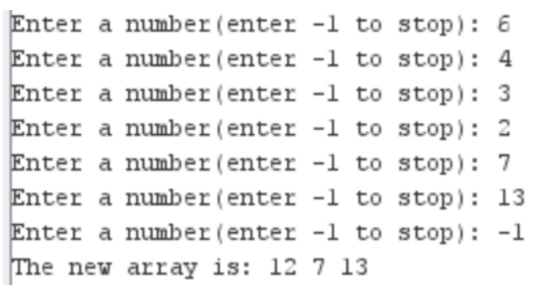
```
Please enter the first coefficient: 6
Please enter the second coefficient: 1
Please enter first number of the sequence: 0
Please enter second number of the sequence: 1
Enter the number you want to calculate (it must be greater than 1): 4
Output: 4. element of the sequence is 23
```

Question 2

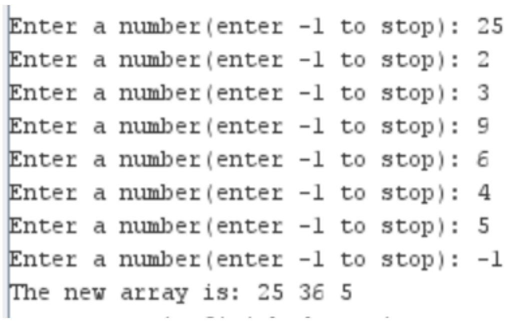
In the 2nd question, we check adjacent numbers in the given sequence, if they are not coprime we delete these two numbers and put their LCM(least common multiple) instead. After we check every adjacent number we display the new array.
the basic flow of the mips code is:

1. Take the number sequence as input from the user
2. Store it in the stack
3. traverse the number sequence
4. check adjacent numbers for coprimes
5. if coprime continue
6. if not coprime, replace the adjacent numbers with their LCM and start comparing again
7. display final number sequence

The code starts with prompting the user to enter the number sequence one by one. to end the sequence the user has to enter the number -1. After that the program starts, from the beginning of the sequence, to take adjacent two numbers and calculates theri gcd using the euclidean algorithm. if their gcd is 1, it means they are coprime, if not they are not. if they are coprime the code proceeds to the next set of adjacent numbers. if not, the program finds the LCM of the two numbers(LCM= a*b/GCD). Replaces the first number with the LCM. Deletes the second number and shifts the number sequence to close the gap created by the deletion. Then it starts to check the numbers, from the beginning of the number sequence. After all numbers are checked the program prints the new sequence by traversing its elements and terminates itself after.
Throughout the code, $s1 and $s2 will be used as global variables to store the GCD and -1 respectively.



```
Enter a number(enter -1 to stop): 6
Enter a number(enter -1 to stop): 4
Enter a number(enter -1 to stop): 3
Enter a number(enter -1 to stop): 2
Enter a number(enter -1 to stop): 7
Enter a number(enter -1 to stop): 13
Enter a number(enter -1 to stop): -1
The new array is: 12 7 13
```



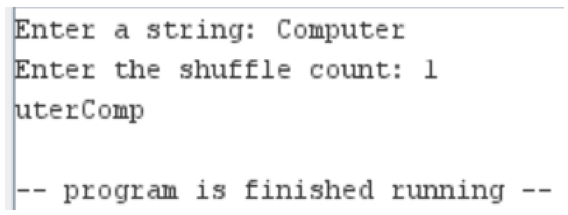
```
Enter a number(enter -1 to stop): 25
Enter a number(enter -1 to stop): 2
Enter a number(enter -1 to stop): 3
Enter a number(enter -1 to stop): 9
Enter a number(enter -1 to stop): 6
Enter a number(enter -1 to stop): 4
Enter a number(enter -1 to stop): 5
Enter a number(enter -1 to stop): -1
The new array is: 25 36 5
```

Question 3

In the 3rd question. We recursively shuffled a string. In order to make it, we first wrote a pseudocode to express the steps that we need to do.

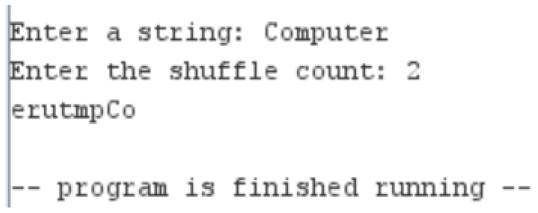
1. Get the inputs from the user.
2. Find the length of the string.
3. Call the shuffle function
4. Swap the first half of the string with the second half.
5. Call the shuffle function for the first half then the second half
6. Return to the 4th step until the shuffle count is 0.

We wrote a strlen procedure to find the length of the input string. After that, We created the shuffle function. In the shuffle function, We get string address, shuffle count and string length as the parameters of the function $a0, $a1 and $a2. We saved the return address and the parameters. Then, we called the swap procedure in the shuffle function. This function basically swaps the two halves of the string. After the swapping finishes. We called the recursive function 2 more times to shuffle all the halves. We are decrementing the shuffle count by 1 in every level of the procedure. If the shuffle count is zero. We basically return to the caller function and finish the shuffle process. Finally, we print the string to the console and exit the program.



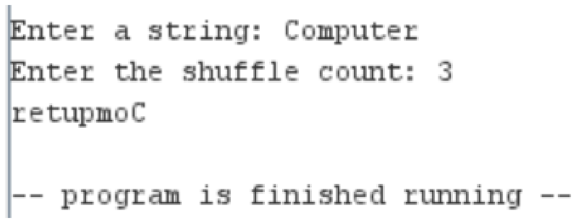
```
Enter a string: Computer
Enter the shuffle count: 1
uterComp

-- program is finished running --
```



```
Enter a string: Computer
Enter the shuffle count: 2
erutmpCo

-- program is finished running --
```



```
Enter a string: Computer
Enter the shuffle count: 3
retupmoC

-- program is finished running --
```

Question 4

In the fourth question, we wrote a MIPS code that iteratively navigates the matrix, which only takes the values 1 and 0, and calls a recursive function on each element. The output is the matrix and the number of 1s in the largest island of the 1-value elements. The steps of this code are as follows:

1. Tour the elements of the Matrix.
2. If the element's value is one, call the checkIsland function.
3. Call checkDown function to check the element below the element.
4. After checking the 1's below, go back to the checkIsland function and call the checkUp function to do the same for checking the elements above.
5. Repeat steps 3 and 4 for each valued element to the right of the element.
6. After all operations are completed, if the tempIslandSize value is greater than maxIslandSize, assign it to the maxIslandSize variable.
7. Apply these operations to each element of the matrix.
8. While navigating the elements of the matrix, print the matrix and finally the maxIslandSize value to the console.

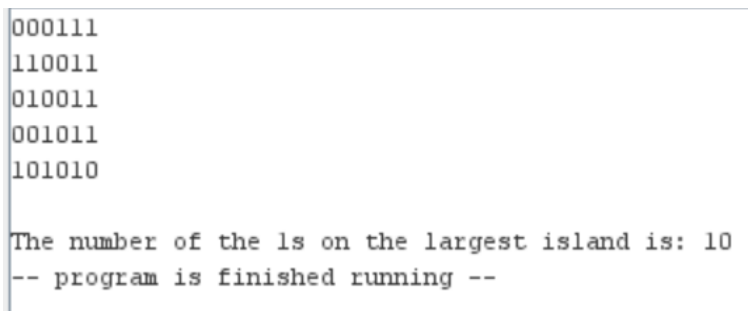
First, $t2(i) and $t3(j) are assigned a value of 0 to be used while navigating the matrix. The number of rows and columns of the matrix is assigned to $t0 and $t1. $t8 is set to maxIslandSize, $a2 is set to 0 to be used for tempIslandSize. $t9 is assigned the starting address of the elements of the matrix.

With nested loops, the matrix begins to be navigated and the element is written to the screen. If its value is 1, the checkIslan function is used. The value i as $a1, tempIslandSize as $a2 and j value as $a3 are sent as parameters to this function. In the function, $a1 and $a3 values are stored in the stack pointer.

Then the checkDown function is called, the same parameters are given to this function and the same parameters are stored. The purpose of the checkDown function is to check the number of elements with a combined value of 1 under the element. It performs this operation recursively. If it is in the last row or the value of the element is 0, the functions end at the beginning. After the checkDown process is completed, the checkUp function is called and performs the same operations for above. After this function, the checkRight function is called. If the value checked on the right is 1, the elements above and below this element are also checked. This process continues until there is no adjacent element on the right, and for each adjacent element with a value of 1, one is added to tempIslandSize.

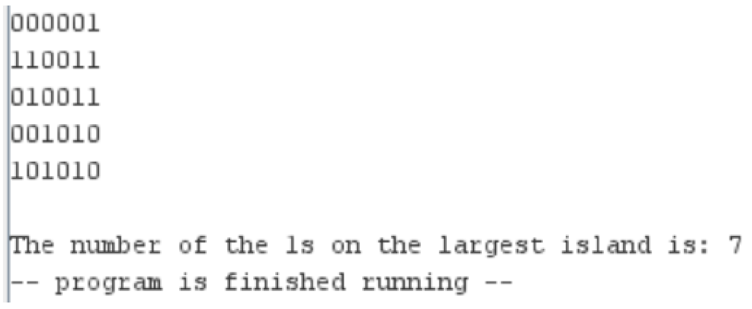
After all the recursive functions are completed, the checkIsland function is returned and the maxIslandSize and tempIslandSize are compared and the resulting tempIslandSize value is assigned to the maxIslandSize variable.

After all elements have been browsed, maxIslandSize is printed to the console.



```
000111
110011
010011
001011
101010

The number of the 1s on the largest island is: 10
-- program is finished running --
```



```
000001
110011
010011
001010
101010

The number of the 1s on the largest island is: 7
-- program is finished running --
```