Natasha Muthoni Pregetter: nmp4: H00326153
Akshay Venkataramana: asv3 : H00324880

Report – CW3 Systems Programming

F28HS: Hardware-Software Interface

4-8-2021

Contents

Functionality (in C)

Variables:

- COL : the amount of colours (constant).
- LEN: the length of the sequences (constant).
- colors: unused variable that would have been used to print colour strings.
- seqlen: the variable used in program to denote sequence length.
- color_names: unused pointer to characters array of colour strings.
- theSeq: unused array for sequence.
- my_array: created integer array to hold sequence.
- guess: integer array to hold user guess.
- almost: the value used to show the user that they have a certain amount of correct values in the wrong positions.
- match: the value used to show the user that they have a certain amount of full matches.

Functions :

- initSeq() : initialises a random sequence of colours.
- arrInit(): sets every value in an array to 1, used as a Boolean.
- showSeq(): shows the generated sequence in the terminal window.
- countMatches(): counts how many matches and almosts the user has in their guess.
- showMatches(): shows the amount of matches and almosts the user got in the terminal.
- readString(): reads the user input into the guess array.
- convertSeq(): convert char array into an int array.
- main(): the main method that debugs and runs the game in the terminal window.

Behaviour of Functions (in C)

initSeq()

This function uses a variable `time_t t;` to ensure that the generator creates a completely random sequence on every program run. It's implemented fully in `srand((unsigned) time(&t));`.
The loop then assigns every available index in my_array to a random variable generated.

arrInit(`int length, int arr[]`)

C does not have a built in boolean value so this is a helper function. It uses a loop to assign every value in an array to 1, which is then used as a substitute for booleans later on in showMatches.
The argument of length is to initialize an array of specific length, and the arr argument is the integer array to be filled with 1s.

showSeq(`int *seq`)

This prints the sequence given in the argument *seq as a line saying “Secret Code ###”

countMatches(`int *seq1, int *seq2`)

Takes in two arguments of two different sequences, one being the users guess, and the other being the randomly generated sequence.
It then creates two integer arrays called *secretary* and *guessarr* using LEN to denote their length, and initialises them using `arrInit(LEN, guessarr);` & `arrInit(LEN, secretarr);.`

The first loop checks both arrays against one another to see how many elements match in the sequences, and if something fully matches it increments the variable matches.
The second loop is used to check if an element is correct, but its in the wrong position. If this is the case it increments the value almost. It has a nested loop to check for the positional difference.

showMatches(`int code,/int *seq1, int *seq2`)

Takes in two sequences and a code that's only used for debugging. It calls the countMatches method with the two sequence arguments and then prints the values of matches and almost to the terminal.

readString(`int *arr`)

Takes in an array that's used to store the user input.
It prompts the user to enter their guess, and then uses the *scanf* to read the user input into the argument *arr*.

convertSeq(`int* arr2, char *arr`)

This takes in an argument of a character array called *arr* and converts it into an integer array called *arr2*.
It does this by using a loop to go through the two arrays and assigning every index in arr2 to a ASCII value of the character in *arr* that has 48 subtracted from it.

main(`int argc, char **argv`)

The function itself is used to access debugging but otherwise it runs the program normally and calls initSeq().

If the player however uses either of the debug methods (-s or -u) we don't run initSeq and instead we manually set the sequence in the case of -s and we set a sequence and a guess in the -u debug case.

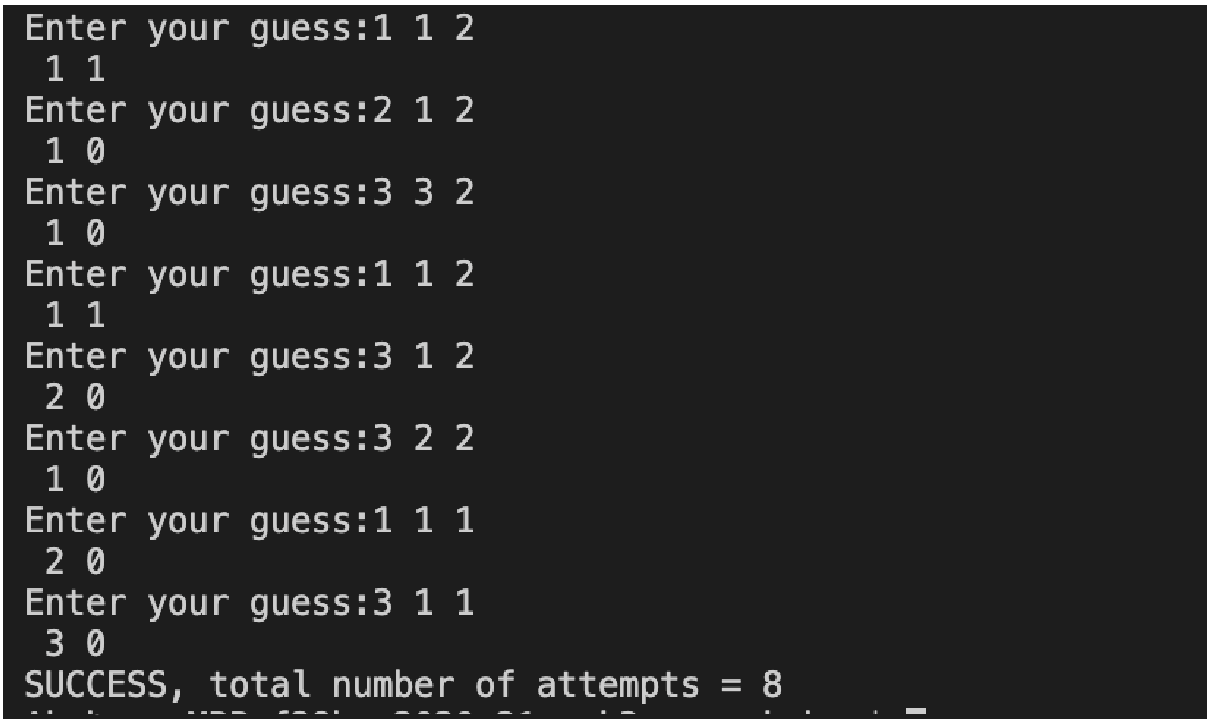
After it calls initSeq the program prints the welcome message to the player and enters the main while loop that continues until either the player finds the sequence or they use up their attempts.
The main loop calls the readString() method to prompt the user for their guess, then calls showMatches(), which counts and shows the matches to the player.
It then increments attempts.

The if, if else, and else, statements afterwards check whether the sequence has been found, which only happens when matches is == 3.

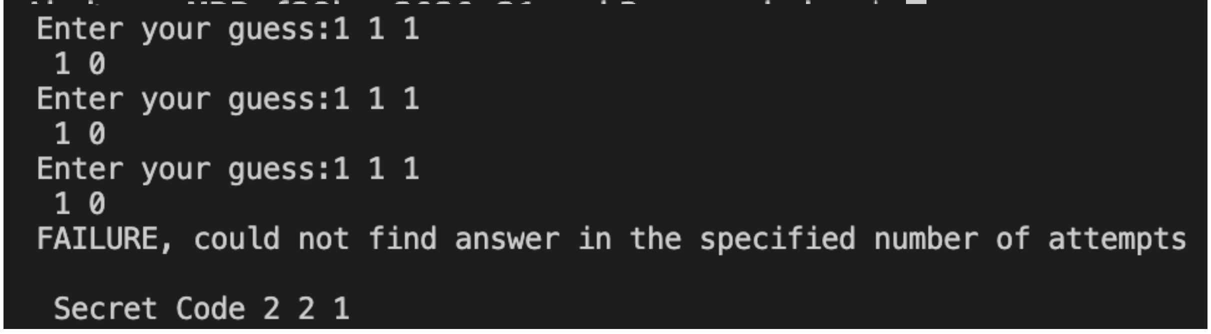
If matches == 3, found is changed to 1, and it exits the while loop.
If the amount of attempts is greater than 9 it also exits out.
Otherwise, it resets the matches and attempts values and goes back to the beginning of the loop.

Outside of the loop are two more conditionals checking if the player found the sequence. It prints either a success or failure message depending on the outcome, and then exits the program.

Sample Execution



```
Enter your guess:1 1 2
 1 1
Enter your guess:2 1 2
 1 0
Enter your guess:3 3 2
 1 0
Enter your guess:1 1 2
 1 1
Enter your guess:3 1 2
 2 0
Enter your guess:3 2 2
 1 0
Enter your guess:1 1 1
 2 0
Enter your guess:3 1 1
 3 0
SUCCESS, total number of attempts = 8
```



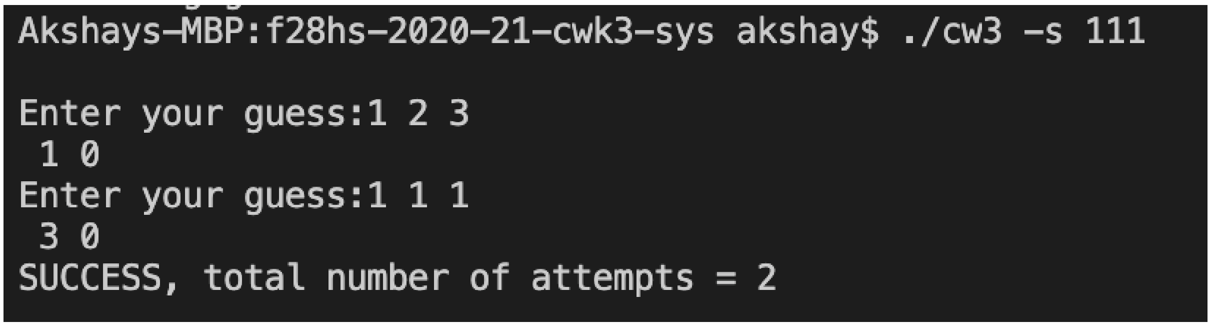
```
Enter your guess:1 1 1
 1 0
Enter your guess:1 1 1
 1 0
Enter your guess:1 1 1
 1 0
FAILURE, could not find answer in the specified number of attempts

 Secret Code 2 2 1
```

Debugging

The debug methods that were implemented for the run of this program were the -s and the -u method.

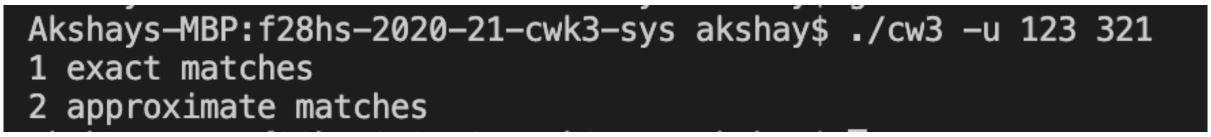
The -s method lets the user set the sequence that they will be guessing. This can be used to debug showSeq(). The -u method is a unit test for the program and the user sets the guess and sequence that they are guessing, which is useful for testing showMatches().



```
Akshays-MBP:f28hs-2020-21-cwk3-sys akshay$ ./cw3 -s 111

Enter your guess:1 2 3
 1 0
Enter your guess:1 1 1
 3 0
SUCCESS, total number of attempts = 2
```

In the picture above we can see the -s debug method running.



```
Akshays-MBP:f28hs-2020-21-cwk3-sys akshay$ ./cw3 -u 123 321
1 exact matches
2 approximate matches
```

In the picture above we can see the -u method running.

This is the implementation of the unit test in our program. This method doesn't work when dealing with sequences that contain duplicates (for example - 331,223). The guess as to why it's unable to pass all the tests is because the approximate never sets the reference values to 0 in the array. Therefore it is unable to form an exact match on either of the 3's in the sequence. This leads to the program adding an extra approximate.
When testing out possible solutions we tried to set the values to 0 in the function where approximates are calculated and it lead to us getting an error for an Abort Trap.

Summary

Functionality of the game itself was a success, however we could not fully debug the program due to a counting error.

Some of the pre-included variables were ignored to make it easier during development, and others were excluded due to time constraints.

The ARM implementation was not completed.