Bryan Wong

805111517

Professor Stahl

CS31

February 10, 2019

Project 3 Report

a. During this project, I ran into several notable obstacles, mostly involving working with strings. I had to be very careful to use the correct index when dealing with the string.at() function, in order to correctly read and compare the letters in the rower string. I also had to be very careful to think of every possible permutation of each of the parameters. As an example, for the time spent rowing parameter, I had to consider each of the following cases: 2 digit minute values, 1 digit minute values, no minute values, minute values greater than 59, second values greater than 59, illegal minute values (0 minutes), etc. Another obstacle was figuring out a way to test one long string for 4 parameters, making sure to test the right part of the initial string for each parameter. I accomplished this by creating 4 separate functions to test separately for the time elapsed, stroke rate, distance, and heart rate. In the isValidRowerString( ) function, I located the indexes of each parameter and fed them to the functions I created.

b. In my program, I created a boolean function that tested separately for each of the four parameters (time elapsed, stroke rate, distance, and heart rate). For each of these functions, a string and the index of the corresponding parameter is given.

For example,

bool validTime(string rowerString, int position)

In my isValidRowerString( ) function, I would take the inputted rowerString, using while( ) loops to “skip” to the indexes of each parameter, feeding them to the according boolean function. For example, I used a while loop to skip any spaces in rowerString, until arriving at the index where the first parameter (time elapsed) should be. The original string and this index are then fed to the validTime( ) function.

With each of the 4 boolean functions, I had to account for any intricacies and permutations of the parameters and code if-else statements accordingly. In the case of validTime( ), I first had to figure out how many spaces in the string were occupied by the parameter, since each different case is treated differently. To do this for the example of time, I had to check for the location of the colon ‘:’ that separates the minutes and times. Knowing that there are only 3 possible configurations for the time elapsed (:XX, X:XX, XX:XX), I used if( ) statements testing for the position of ‘:’. Once the number of digits occupied by the minutes was known, I tested to make sure that the actual time values are valid. This was accomplished by using if statements with inequality expressions inside. Since s.at( ) represents some char value with a corresponding integer value, I can test to make sure that the char value lies between some range of numbers. For example, the expression (rowerString[position] >= 48 && rowerString[position] <= 57) tests that the char value of rowerString at the index position is from 0-9.

Here is pseudocode for validTime( ):

if the character of rowerString at index position is a colon

check to make sure that the values of :XX are in their appropriate bounds (:00 - :59) and return true/false accordingly

otherwise, if the character of rowerString at index position + 1 is a colon

check to make sure that the values of X:XX are in their appropriate bounds (1:00 - 9:59) and return true/false accordingly

otherwise, if the character of rowerString at index position + 2 is a colon

check to make sure that the values of XX:XX are in their appropriate bounds (10:00 - 59:59) and return true/false accordingly

otherwise, if there is no colon located in the first 3 digits, return false

Once this function returns true/false, the isValidRowerString( ) will read it and respond accordingly. If any parameter is tested as invalid, isValidRowerString( ) will return false. If the parameter is tested valid, isValidRowerString( ) will locate the next parameter using the aforementioned while loop method, and feed it to the next function testing for that particular parameter. Each of these functions has certain intricacies for its parameter that must be accounted for. For the stroke rate, values can be 1-3 digits, and must be greater than 0. For distance, a while loop must be implemented because values can be an infinitely long positive value. For heart rate, values can be 1-3 digits and must be greater than 0. I also had to be careful to account for extraneous characters after the parameters that would render the string invalid. Once every parameter has been tested and deemed valid, isValidRowerString( ) will finally return true.

For the int functions that return the values of a certain parameter ( int totalDistance( ), int heartRate( ), int strokesPerMinute( ), etc.), each function will first check that the inputted string passes isValidRowerString( ). If the string doesn’t pass, then the function will return -1. If the string passes, the function will use while loops to skip to the location of the parameter. Since we know that the rowerString is formatted correctly, we can use certain elements of the string (such as the ’s/m’ or ‘m’) to skip to. The function will then read through the parameter and return it as an integer value. Since the input is a string with char values, we must subtract the char value of the digit by the char value of ‘0’ to get the integer equivalent. For multi-digit parameters, we must multiply the earlier digits by 10 and then add them to the later digits to get the integer equivalent. Once the end of the parameter is reached, the integer value of it is returned.

Here is the pseudocode for heartRate( ):

int heartRate(string rowerString)

if rowerString is a valid rower string, then continue. Otherwise, return -1.

set the index pointer to 0.

while the value of the string at the index pointer is not ‘m’, increment the index pointer. (skip to the first m)

increment the index pointer by 1 (move the pointer to the space after the first m)

while the value of the string at the index pointer is not ‘m’, increment the index pointer. (skip to the second m)

increment the index pointer by 1 (move the pointer to the space after the second m)

while the value of the string at the index pointer is a space, increment the index pointer. (skip to the next nonspace digit. This will be the index of the first digit for the heart rate parameter)

check the size of the parameter:

if the heart rate is one digit, return the int equivalent of that digit, calculated by subtracting the char value of ‘0’ from the char value of the digit.

if it is two digits, multiply the int value of the first digit by 10 and add it to the int value of the second digit.

if it is three digits, multiply the int value of the first digit by 100, add it to the int value of the second digit times 10, and add it to the int value of the third digit.

c. Here is a list of several test cases I used to test my code:

// *Given Test Cases*

**isValidRowerString( ) should be valid:**

assert(isValidRowerString(":14 28 s/m 42 m 110")); - No minutes digits

assert(isValidRowerString(" :14 28 s/m 42 m 110")); - Large amount of spaces preceding first parameter

assert(isValidRowerString("1:14 28 s/m 42 m 110")); - One minutes digit

assert(isValidRowerString("59:14 28 s/m 42 m 110")); - Max minute value of 59

assert(isValidRowerString("59:59 28 s/m 42 m 110")); - Max seconds value of 59

assert(isValidRowerString("59:59 999 s/m 42 m 110")); - Max stroke rate value of 999

assert(isValidRowerString("59:59 999 s/m 42 m 999")); - Max heart rate value of 999

**isValidRowerString( ) should be invalid:**

assert( ! isValidRowerString("asdf:14 28 s/m 42 m 110")); - Invalid minutes value

assert( ! isValidRowerString(":14 28 s/m 42 m 110 asdf")); - Extra characters after heart rate

assert( ! isValidRowerString(":14 28 s/m 42 m 110")); - Extra spaces before “s/m”

assert( ! isValidRowerString("0:14 28 s/m 42 m 110")); - Minutes value of 0

assert( ! isValidRowerString("0000:14 00028 s/m 00042 m 000110")); - Extra leading zeroes

assert( ! isValidRowerString(":-14 -28 s/m -42 m -110")); - Negative values

assert( ! isValidRowerString("555:14 28 s/m 42 m 110")); - Minutes value over the max

assert( ! isValidRowerString(":555 28 s/m 42 m 110")); - Seconds value over the max

assert( ! isValidRowerString(":14 99123 s/m 42 m 110")); - Stroke rate value over the max

assert( ! isValidRowerString(":14 0 s/m 42 m 110")); - Stroke rate of 0

assert( ! isValidRowerString(":14 28 s/m 42 m 99123")); - Heart rate over the max

assert( ! isValidRowerString(":14 28 s/m 42 m 0")); - Heart rate of 0

*// Self Test Cases*

**isValidRowerString( ) should be valid:**

assert(isValidRowerString("11:11 1 s/m 1 m 1")); - Extra spaces before stroke rate

assert(isValidRowerString("11:11 1 s/m 1 m 1")); - Extra spaces before distance

assert(isValidRowerString("11:11 1 s/m 1 m 1")); - Extra spaces before heart rate

assert(isValidRowerString(":00 1 s/m 1 m 1")); - Time value of :00

assert(isValidRowerString(":00 1 s/m 9999 m 1")); - Large distance value

assert(isValidRowerString(":00 1 s/m 9999999 m 1")); - Very large distance value

assert(isValidRowerString("59:59 999 s/m 999999 m 999")); - Maxed values and very large distance value

assert(isValidRowerString(":59 999 s/m 999999 m 999")); - Max seconds and very large distance value

**isValidRowerString( ) should be invalid:**

assert( ! isValidRowerString("11:11 1 s/m 0 m 1")); - Distance value of 0 is invalid

assert( ! isValidRowerString(":11 1 s/m 1 m 1 ")); - Extra spaces after heart rate are invalid

assert( ! isValidRowerString(":59 9999 s/m 999999 m 999")); - Stroke rate over max

assert( ! isValidRowerString(":59 999 s/m 999999 m 9999")); - Heart rate over max

assert( ! isValidRowerString("11:11 1 s /m 1 m 1")); - Extra space between s and /

assert( ! isValidRowerString("11: 11 1 s/m 1 m 1")); - Extra space between : and seconds value

assert( ! isValidRowerString("1 s/m 1 m 1")); - Missing time value

assert( ! isValidRowerString("11:11 1 s/m 1")); - Missing distance value

assert( ! isValidRowerString("11:11 1 m 1")); - Missing stroke rate value

assert( ! isValidRowerString("11:11 1 s/m 1 m")); - Missing heart rate

assert( ! isValidRowerString("11:11 1 s/m 1 1")); - Missing m in distance

assert( ! isValidRowerString("11:11 1 s/m 1 m -1")); - Negative heart rate

assert( ! isValidRowerString("11:11 1 s/m 1 m 0")); - Heart rate of 0 is invalid

assert( ! isValidRowerString("11:11 1 s/m 09 m 1")); - Leading zeroes in distance

assert( ! isValidRowerString("11:11 1 s/m -1 m 1")); - Negative distance

assert( ! isValidRowerString("11:11 -1 s/m 1 m 1")); - Negative stroke rate

assert( ! isValidRowerString("11:11 1 m 1 m 1 ")); - Missing s/

assert( ! isValidRowerString("11:11 01 s/m 1 m 1")); - Leading zeroes in stroke rate

assert( ! isValidRowerString("11:11 1000 s/m 1 m 1")); - Stroke rate larger than max

assert( ! isValidRowerString("11:11 1 s m 1 m 1")); - Missing /

assert( ! isValidRowerString("11:11 1 s/ 1 m 1")); - Missing m in stroke rate

assert( ! isValidRowerString("11:11 0 s/m 1 m 1")); - Stroke rate of 0 is invalid

assert( ! isValidRowerString("11:11 1s/m 1 m 1")); - No space between stroke rate value and s/m

assert( ! isValidRowerString(":: 1 s/m 1 m 1")); - Two ‘:’ instead of time

assert( ! isValidRowerString(":111 1 s/m 1 m 1")); - Extra seconds digit

assert( ! isValidRowerString("111:12 1 s/m 1 m 1")); - Extra minutes digit

assert( ! isValidRowerString("1 s/m 1 m 1")); - Missing time

assert( ! isValidRowerString(": 1 s/m 1 m 1")); - Only apostrophe in time

assert( ! isValidRowerString(":60 1 s/m 1 m 1")); - Seconds digit one greater than max

assert( ! isValidRowerString("60:00 1 s/m 1 m 1")); - Minutes digit one greater than max

assert( ! isValidRowerString("00:00 1 s/m 1 m 1")); - Zeroes in minutes digits

assert( ! isValidRowerString("0:00 1 s/m 1 m 1")); - Zeroes in minutes digit

assert( ! isValidRowerString("9:0a 56 s/m 42 m 110")); - invalid characters in time

assert( ! isValidRowerString("9:00 5a6 s/m 42 m 110")); - invalid characters in stroke rate

assert( ! isValidRowerString("9:00 56 s/m 4a2 m 110")); - invalid characters in distance

assert( ! isValidRowerString("9:00 56 s/m 42 m 1a0")); - invalid characters in heart rate

assert( ! isValidRowerString("0000:14 28 s/m 42 m 110")); - leading zeroes in time

assert( ! isValidRowerString("00:14 00028 s/m 42 m 110")); - leading zeroes in stroke rate

assert( ! isValidRowerString("00:14 28 s/m 00042 m 110")); - leading zeroes in distance

assert( ! isValidRowerString("00:14 28 s/m 42 m 00110")); - leading zeroes in heart rate

assert( ! isValidRowerString("11:1 1 s/m 1 m 1")); - only one seconds digit

assert( ! isValidRowerString("11:11 1 s/m 1 m 1")); - Extra spaces between stroke rate and s/m

assert( ! isValidRowerString("11:11 1 s/m 1 m 1")); - Extra spaces between distance and m

assert( ! isValidRowerString("14 28 s/m 42 m 110")); - Missing :

assert( ! isValidRowerString("10:14 28 42 m 110")); - Missing s/m

assert( ! isValidRowerString("10:14 28 s/m 42 110")); - Missing m

assert( ! isValidRowerString("28 s/m 10:14 42 m 110")); - Parameters out of order

assert( ! isValidRowerString("")); - Empty string

assert( ! isValidRowerString(" ")); - Space