Project 3 Report

Melody Chen

#705120273

**Obstacles faced:**

One of the obstacles that I first faced when I started working on project 3 was forgetting that there could not be extra spaces between the minute elapsed and seconds elapsed. My program tested for finding the first number, then finding a colon, and then finding another number. The mistake I made with this logic was not taking into account that all three of these components have to be right next to each other. I allowed for strings such as “50 :56” and “50: 56”, which are noted as invalid strings by the specs of Project 3. I quickly fixed this problem by adding more conditions to make sure that the colon is between the two numbers without extra spaces.

**Design of Program:**

The design of my program is based on the specs provided for Project 3 and centers upon the function bool isValidRowerString(string rowerString); . This function uses other functions in order to produce its desired output. See below for the pseudo code for this function and other function it implements:

isValidRowerString(string rowerString)

Initialize current position to 0

Tests if minute value is valid

Test if there is semicolon in the right place and number

Test if there is not a minute value, then if second value in the right place with and number

Test if first number fits the specs

Tests if seconds value is valid

Tests if number fits the spec

Tests if strokesPerMinute is valid

Tests if strokesPerMinute is a positive non-zero value less than 1000

Tests if “s/m” is in the indicated location after value of strokesPerMinute

Update current location to after “s/m”

Tests if totalDistance is valid

Tests if totalDistance is a positive non-zero value

Tests if “m” is in the indicated location after value of totalDistance

Update current location to after “m”

Tests if heartRate is valid

Tests if heartRate is a positive non-zero value less than 1000

Tests if there are extra ending character or space

returns true if makes through all the conditions

int elapsedMinutes(string rowerString)

test if rowerString is valid

get and return the elapsedMinutes

int elapsedSeconds(string rowerString)

tests if rowerString is valid

get and return elapsedSeconds

int strokesPerMinute(string rowerString)

tests if rowerString is valid

get and return strokesPerMinute

int totalDistance(string rowerString)

tests if rowerString is valid

get and return totalDistance

int heartRate(string rowerString)

tests if rowerString is valid

get and return heartRate

int getIndex(string s, string str, int index)

loops through string s to look for str

return -1 if not found

return index if found

int number(string s, int& startingPosition)

loops through all the spaces

finds the first number

sum up values of the number

returns number

**List of test cases and explanations:**

assert(isValidRowerString(" 5:00 500 s/m 500 m 600")==**true**);*//tests for valid*

assert(isValidRowerString(" 5:00")==**false**);*//missing data*

assert(isValidRowerString(" 5:00 500 s/m ")==**false**); *//missing data*

assert(isValidRowerString(" 5:00 500 s/m 500 m")==**false**); *//missing data*

assert(isValidRowerString(" 5:5 500 s/m 500 m 600")==**false**); *//extra seconds value*

assert(isValidRowerString(" 05:50 500 s/m 500 m 600")==**false**); *//leading zero*

assert(isValidRowerString(" 5: 50 500 s/m 500 m 600")==**false**); *//extra seconds value*

assert(isValidRowerString(" 000005:50 500 s/m 500 m 600")==**false**); *//leading zero*

assert(isValidRowerString(" :50 500 s/m 500 m 600")==**true**);*//tests for no minute value*

assert(isValidRowerString(" :60 500 s/m 500 m 600")==**false**); *//invalid seconds value*

assert(isValidRowerString(" 5 :59 500 s/m 500 m 600")==**false**); *//extra spaces*

assert(isValidRowerString(" 59: 500 s/m 500 m 600")==**false**); *//missing seconds value*

assert(isValidRowerString(" 0:50 500 s/m 500 m 600")==**false**); *//invalid minute value*

assert(isValidRowerString(" 50:0 500 s/m 500 m 600")==**false**); *//single 0 value for seconds*

assert(isValidRowerString(" 555:50 500 s/m 500 m 600")==**false**); *//invalid minute value*

assert(isValidRowerString(" a60:50 500 s/m 500 m 600")==**false**); *//invalid leading character*

assert(isValidRowerString(" 5:a50 500 s/m 500 m 600")==**false**); *//invalid character in string*

assert(isValidRowerString(" ::50 500 s/m 500 m 600")==**false**); *//extra colon*

assert(isValidRowerString(" 5:050 500 s/m 500 m 600")==**false**); *//leading zero for seconds value*

assert(isValidRowerString(" 5:05 500 s/m 500 m 600")==**false**); *//extra spaces*

assert(isValidRowerString(" 5:05 500 00 s/m 500 m 600")==**false**); *//extra zeroes*

assert(isValidRowerString(" 5:05 50s/m 500 m 600")==**false**); *//missing spaces*

assert(isValidRowerString(" 5:05 500 s/m 500 m 600")==**true**); *//tests for validity*

assert(isValidRowerString(" 5:05 1000 s/m 500 m 600")==**false**); *//s/m value too large*

assert(isValidRowerString(" 5:05 0500 s/m 500 m 600")==**false**); *//extra leading zeroes*

assert(isValidRowerString(" 5:05 499 s/m 500 m 600")==**true**);

assert(isValidRowerString(" 5:05 500 ssm 500 m 600")==**false**); *//wrong spelling*

assert(isValidRowerString(" 5:05 500 s /m 500 m 600")==**false**); *//extra spaces*

assert(isValidRowerString(" 5:05 100 s/ m 500 m 600")==**false**); *//extra spaces*

assert(isValidRowerString(" 5:05 100 s 500 m 600")==**false**); *//wrong spelling*

assert(isValidRowerString(" 5:00 500 s/m 0 m 600")==**false**); *//zero for distance rowed*

assert(isValidRowerString(" 5:00 500 s/m 50m 600")==**false**); *//missing space*

assert(isValidRowerString(" 5:00 500 s/m -50 m 600")==**false**); *//negative values*

assert(isValidRowerString(" 5:00 500 s/m 50 m 600")==**false**); *//extra space*

assert(isValidRowerString(" 5:00 500 s/m 50 s 600")==**false**); *//wrong spelling*

assert(isValidRowerString(" 5:00 500 s/m 1000 m 600")==**true**); *//can handle large distance rowed*

assert(isValidRowerString(" 5:00 500 s/m 50 m 600")==**true**); *//can handle extra spaces*

assert(isValidRowerString(" 5:00 500 s/m 50 00 m 600")==**false**); *//extra zeroes*

assert(isValidRowerString(" 5:00 500 s/m 500 m 1000")==**false**); *//heartRate value invalid*

assert(isValidRowerString(" 5:00 500 s/m 500 m 0")==**false**); *//heartRate value cannot be zero*

assert(isValidRowerString(" 5:00 500 s/m 500 m -500")==**false**); *//negative heartRate*

assert(isValidRowerString(" 5:00 500 s/m 500 m 50 50")==**false**); *//extra numbers*

assert(isValidRowerString(" 5:00 500 s/m 500 m a50")==**false**); *//extra char*

assert(isValidRowerString(" 5:00 500 s/m 500 m 50abc")==**false**); *//extra chars*

assert(elapsedSeconds(" 5:00 400 s/m 50 m 1999")==-1); *//unc return -1 for invalid rowerString*

assert(elapsedMinutes(" 50:00 400 s/m 70 m 199")==50); *//func returns correct min value*

assert(elapsedSeconds(" 50:00 400 s/m 70 m 199")==0); *//function reuturns correct seconds value*

assert(strokesPerMinute(" 50:00 400 s/m 70 m 199")==400); *//function returns correct s/m value*

assert(totalDistance(" 50:00 400 s/m 70 m 199")==70); *//function returns correct distance rowed*

assert(heartRate(" 50:00 400 s/m 70 m 199")==199); *//function returns correct heartRate*