HW: Formula 1

Overview

Objectives

- Work with one-dimensional arrays
    - Traversing
    - Accessing
    - Staying within array bounds
    - Initializing
    - Sentinel values (c string)
- Work with parallel arrays
- Validation of data

Submission

The source files to submit to Gradescope are named:

1. `rank.cpp`
2. `rank_functions.cpp`
3. `rank_functions.h`

Introduction

The Circuit of the Americas will host several Formula 1 automotive racing events including the United States Grand Prix. We have been asked to write software to determine the results when 9 cars race. We are to code the software with C++ and use parallel **arrays (not vectors)**.

Example Run of the Race Result

```
[1]  32.70 Moore              (USA)  +0.00
[2]  33.40 Munson             (TKY)  +0.70
[3]  36.50 Polsley            (RUS)  +3.80
[4]  38.00 Reardon            (ARG)  +5.30
[5]  45.80 Taele              (ENG)  +13.10
[6]  50.10 Darlington         (ICE)  +17.40
[7]  52.34 Nemec              (CHN)  +19.64
[8]  60.34 Da Silva           (NIC)  +27.64
[9]  76.45 Lupoli             (ITY)  +43.75
```

The output shown above is formatted and the function for that display is given. You will not have to make any edits or changes.

The program will read in data in the format as shown below.

```
32.7 USA 12 Moore
36.5 RUS 35 Polsley
45.8 ENG 73 Taele
52.34 CHN 14 Nemec
76.45 ITY 23 Lupoli
33.4 TKY 82 Munson
38.0 ARG 88 Reardon
50.1 ICE 41 Darlington
60.34 NIC 50 Da Silva
```

Notice the data will come in this form and order on each line:

- Time Completed
- Country
- Car number
- Driver's Last name

Test Data

To help with debugging, several files are provided containing some test data. With input redirection, we can run the program and provide the file rather than manually typing in all of the values.

Your program will pull the data from standard input (e.g. cin) and place each piece of data into separate but parallel arrays:

```
          [0]   [1]     [2]   [3]   [4]     [5]    [6]     [7]         [8]

time      32.7  36.5    45.8  52.34 76.45   33.4   38      50.1        60.34

country   USA   RUS     ENG   CHN   ITY     TKY    ARG     ICE         NIC

number    12    35      73    14    23      82     88      41          50

lastname  Moore Polsley Taele Nemec Lupoli  Munson Reardon Darlington  Da Silva
```

Using the data in the parallel arrays, you will rank each driver based on their time from low to high.

- *There is absolutely **NO ORDERING** or **SORTING** of the data.*
- *You put each driver's rank into a parallel array which will be used to print the results in the correct order.*

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
time	32.7	36.5	45.8	52.34	76.45	33.4	38	50.1	60.34
country	USA	RUS	ENG	CHN	ITY	TKY	ARG	ICE	NIC
number	12	35	73	14	23	82	88	41	50
lastname	Moore	Polsley	Taele	Nemec	Lupoli	Munson	Reardon	Darlington	Da Silva
rank	**1**	**3**	**5**	**7**	**9**	**2**	**4**	**6**	**8**

Think of the rank function as finding the lowest number first, marking the appropriate rank array index with 1, then continuing to find the next value as long as the rank is not already determined. It will need to check the rank position to see if it has been marked and not consider those.

Roadmap

This is a guide on how to tackle doing this homework. Specific requirements for functions are in the Requirements section below.

1. Get the starter code and test files (there is a zip file in there if you want to get everything at once)
    a. Code Files
        - `rank.cpp`
        - `rank_functions.h` (you should read this file)
        - `rank_functions.cpp`
    b. Test Files
        - Various text files to help you test your code (.txt)
    c. **Read the header file!** It gives details about each function you will implement. The source files (.cpp files) have comments marked // TODO where you will need to write code.
2. Review the code.
    a. Compile and run it before making any changes.
        - It won't do anything but it also won't crash.
            - So **all** errors were introduced by you!

b. Read the files

- Read the comments for more information.
  - They give details about each function you will implement.
- Header file
  - You are required to implement all functions listed in rank_functions.h except for the print_result function.
- Source files
  - The source files (.cpp files) have comments marked // TODO where you will need to write code.
  - DO NOT MAKE CHANGES TO print_function!

3. Implement functions

a. **Write your code so that it is easy to understand!**

- Conventions like meaningful variable names and commenting will make it easier for you to understand your own code and implement your algorithms!
  - This also helps if you need to get help from a TA, Peer Teacher, or instructor.
- Use descriptive (long) naming conventions for variables and functions.
- **Add comments** to the code to describe anything which is not obvious from the code.
- Use (indentation, blank lines) to visually organize code.
- Use functions to reduce code duplication and increase abstraction.

b. Write a little bit of code, test and debug it. Repeat until finished.

- Writing all of your code and then going back to debug can double, triple, quadruple or more the time it takes you to implement your solution.

c. You should attempt to implement functions based on their dependencies.

- Since get_driver_data depends on having initialized arrays, you should do all of the **prep_*_array**[1] functions before doing get_driver_data.
- Since the set_rankings function relies on arrays populated with data, you should do **get_driver_data** next.
- Finally, you should do **set_rankings** next since print_results depends on it.

---

[1] * is used as a wildcard entry in CS literature. This is used to represent any character. For example, prep_*_array would mean using prep_string_array, prep_double_array, prep_unsigned_int_array ... etc.

d. Recompile and rerun after completing each function.
    - Note: You can test and debug before you add all functionality to a function. Get one aspect to work and then add functionality in measured increments until you complete the full functionality.
    - Check for errors.
    - If no errors, move on
    - Else, start debugging
e. Once you have some functionality, submit to Gradescope.
    - If the basic tests for that function pass, move on
    - Else, start debugging
f. Continue by picking new tests and writing just enough code to pass them, adding more functionality each time.

Useful Functions

- **>>**
    - gets a value by skipping over whitespace and stopping when it gets to whitespace (i.e. a space, at tab, new line, etc.)
- **cin.getline(char[], STRING_SIZE)**
    - Available by including cstring
    - Gets all characters from the current location until the end of the line including whitespace.
- **isupper(char)**
- **isalpha(char)**
- **isspace(char)**
- **strlen(const char[])**
    - Available by including cstring
- **strcpy(char[], char[])**
    - Available by including cstring
    - Use this to assign a cstring to another cstring since using the assignment operator (‘=’) does not work with cstrings.
        - Assuming a and b are cstrings and assigning a to b:

- Correct: `strcpy(a, b);`
- Incorrect: `a = b;`

Requirements

Allowed Includes

- `<iostream>`
- `<cstring>`
    - Not to be confused with `<string>`. `<string>` is forbidden for this assignment.
- `<iomanip>`
- `"rank_functions.h"`

Functions

main function

- Create arrays for time, country, driver number, name, and rank.
    - They should use the SIZE constant in rank_functions.h for their size.
- The program should receive information from cin.
    - You can use Linux Input Redirection to avoid typing manually.
- The program should output “Bad input” to standard out if the program fails to successfully load the driver data.

Prep functions

These functions initialize all of the values in each array before they are used.

1. `prep_double_array` elements should all be set to `0.0`
2. `prep_unsigned_int_array` elements should all be set to `0`
3. `prep_string_array` elements should all be set to “N/A”
    - Don’t forget that all c-strings must end with a ‘\0’

trim

This function removes whitespace from the beginning and end of a cstring.

- Any whitespace in the middle will be preserved and left alone.
- Note that you should end with a cstring. So if it is an empty string, it should still have '\0' at index 0.

get_driver_data

This function loads data from standard in (e.g. cin) into the parallel arrays. You can only process valid data. If any data is invalid, the function returns false.

Data validation:

- The double containing **time** information:
    - Be a non-zero positive number.
- The string containing **country** information:
    - Contains only capital letters 'A' - 'Z'
    - Contains exactly 3 characters
        - Note that an empty string is not valid
- The unsigned int **number** information:
    - Contains 1 or 2 digits
- The string containing **name** information:
    - Contains only
        - alphabet characters 'A' - 'Z' and 'a' - 'z'
        - space character ' ' between parts of a last name like Da Silva
        - You should trim whitespace from the beginning and end before checking for valid characters.
    - Contains more than one character
        - Note that an empty string is not valid
        - You should trim whitespace from the beginning and end before checking if it is long enough.

set_rankings

This function looks at the time in each element of the parallel array. The corresponding element in the rank array is assigned the appropriate rank. The fastest time's rank is one and the slowest time's rank is 9.

Linux Input Redirection

As we did in the Grade calculator homework, we can use input redirection to have our program treat data from a text file as if it were coming from standard input (i.e. cin). You can use the less than symbol '<' to direct data to your program. For example

```
$ g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ *.cpp -o main
$ ./main < good_data01.txt
Final results!!
[1]  20.67 Fries                (ISL)  +0.00
[2]  24.29 Maines               (HKG)  +3.62
[3]  26.84 Wilkey               (CAY)  +6.18
[4]  34.32 Fiorentino           (MDV)  +13.66
[5]  39.67 Sheldon              (GUM)  +19.00
[6]  44.28 Weed                 (CZE)  +23.61
[7]  46.14 Payton               (RSA)  +25.47
[8]  66.91 Shipp                (CRC)  +46.25
[9]  85.30 Nagel                (MAR)  +64.63
```