Homework: Taxi Trips

# Overview

Many cities make available to the public data on transportation services such as bike-sharing, taxis, buses, and subways. The datasets can be useful for several purposes, including identifying patterns and trends that can help predict the demand for transportation systems.

This homework works with taxi trip data. Your program needs to read a file with information on taxi trips and output a summary of the data.

# Input data format

For each taxi ride, the file contains the following information:

taxi\_id trip\_seconds trip\_miles trip\_fare trip\_tip payment\_type

where:

* taxi\_id: a string that identifies the taxi providing the ride
* trip\_seconds: the ride duration in seconds
* trip\_miles: the driven distance in miles, when available. If not available, the number is reported as zero.
* trip\_fare: the cost of the trip
* trip\_tip: the tip provided by the customer
* payment\_type: a string with one of the following values:   
  “CreditCard”, “Cash”, “NoCharge”, “Unknown”

The input files are “cleaned up” by the government agency providing the data, but they still may present errors. In this homework, you are expected to handle **only** the following invalid scenarios:

* Values for trip\_seconds that exceed 24 hours.
* Values for trip\_miles that exceed 1500 miles.
* Values for trip\_fare + trip\_tip that exceed 5000 dollars.
* Value for payment\_type that is not “CreditCard”, “Cash”, “NoCharge”, or “Unknown”.
* Values for trip\_seconds and trip\_miles are both zero but trip\_fare is not zero.

Any trip containing invalid data must be ignored, i.e., they are not included in the calculators for the output report. Again, you **do not** need to cover other invalid input scenarios, such as the file containing a non-number text when a number is expected.

The maximum number of rides in an input file is MAX\_TRIPS (defined in the starter code as a global constant in taxi\_trips.h)

An example of an input file

Taxi\_0 660 1.1 9.05 4 CreditCard

Taxi\_1 300 1.0 5.25 0 Cash

Taxi\_2 540 1.1 800.01 4200 Cash

Taxi\_3 360 1.85 5.45 2 CreditCard

Taxi\_4 180 1.1 4.65 0 Cash

Taxi\_5 60 1.0 3.25 0 Cash

Taxi\_2 300 1.8 6.05 3.95 Cash

Taxi\_7 840 2.8 10.45 3 Cash

Taxi\_8 1080 9.3 21.05 5 Cash

Taxi\_1 600 0.9 4.25 0.75 Unknown

Taxi\_21 3600 40 121.11 20 CreditCard

In the example listed above, the file contains information from several rides. The first one, by taxi Taxi\_0, has a duration of 660 seconds, distance of 1.1 miles, fare cost of $9.05, tip of $4, and it was paid by credit card. The third ride in the file, by Taxi\_2, is invalid because the fare + tip exceeds 5000 dollars.

# Output data format

Your program needs to generate as output a report with the following information:

* Number of trips with valid data
* Number of trips with invalid data
* Maximum, minimum, average, and mode[[1]](#footnote-0) for trip duration, distance, fare, and tip
* Length of longest sequence of trips without a tip

The data must be reported with the format in the example below.

## An example of output data

For the input file example above, the program would write to standard output the following:

Number of trips: 10

Number of trips with invalid data: 1

Duration: max 3600 min 60 avg 798 mode 300

Distance: max 40 min 0.9 avg 6.085 mode 1.1

Fare: max 121.11 min 3.25 avg 19.056 mode 9.05

Tip: max 20 min 0 avg 3.87 mode 0

Length of longest sequence without tips: 2

# Objectives

1. Practice using C++ file input in a simple scenario. You will work with more complex data validation scenarios in upcoming labwork and homework assignments.
2. Initial practice with:
   1. Using C++ arrays to store data
   2. Writing functions that receive arrays as arguments
   3. Writing code that accesses elements of an array

# Starter Code

The starter code on Mimir contains three files:

1. main.cpp := contains the user interface that drives the program
2. taxi\_trips.h := contains required function declarations, read it.
3. taxi\_trips.cpp := contains function definitions. This is where most of your work gets done.

# Requirements

1. The source files to submit are named
   1. main.cpp
   2. taxi\_trips.h
   3. taxi\_trips.cpp
2. You are required to implement the following functions, which are declared and described in the taxi\_trips.h header file, **which you should read**.  
   1. **bool is\_payment\_valid(string type, string valid\_types[],**

**unsigned int size\_valid\_types)**

Assuming that array payments holds the values  
 {“CreditCard”, “Check”, “Cash”, “Bitcoin”}

is\_payment\_valid(“Cash”, payments, 4) returns true

is\_payment\_valid(“GiftCertificate”, payments, 4)   
 returns false

is\_payment\_valid(“Bitcoin”, payments, 3)   
 returns false

is\_payment\_valid(“Cash”, payments, 0) should throw  
 std::invalid\_argument

* 1. **double max(double values[], unsigned int size)**

Assuming that the array\_a holds the values {1, 0, 2, 100}

max(array\_a, 4)should return 100

max(array\_a, 2)should return 1

max(array\_a, 0) should throw std::invalid\_argument

* 1. **double min(double values[], unsigned int size)**

Assuming that the array\_a holds the values {1, 0, 2, 100}

min(array\_a, 4)should return 0

min(array\_a, 1)should return 1

min(array\_a, 0) should throw std::invalid\_argument

* 1. **double average(double values[], unsigned int size)**Assuming that the array\_a holds the values {10, 40, 16}

average(array\_a, 3) should return 22

average(array\_a, 0) should throw std::invalid\_argument

* 1. **double mode(double values[], unsigned int size)**Assuming that the array\_a holds the values {10, 10, 16} and array\_b holds {10, 16, 10, 16, 16}:

mode(array\_a, 3) should return 10

mode(array\_b, 5) should return 16  
 mode(array\_b, 4) should return 10  
 mode(array\_a, 0) should throw std::invalid\_argument

* 1. **unsigned int length\_longest\_seq\_zeros(double values[],**

**unsigned int size)**

Assuming that the array\_a holds the values   
 {10, 0, 0, 0.8, 0, 0, 0, 40, 0, 16}  
 length\_longest\_seq\_zeros(array\_a, 10) should return 3  
 For size zero, the function should throw

std::invalid\_argument

1. You may use the following includes:
   1. iostream
   2. ifstream
   3. limits
   4. sstream
   5. string
   6. taxi\_trips.h
2. The program must compile without warnings or errors.

g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ taxi\_trips.cpp main.cpp

1. The program must run without errors or unhandled exceptions.

(user input in **bold red**; everything else is output)

⎵ is a space character (‘ ’)

⮒ is a newline character (‘\n’, also displayed as ↵)

$⎵./a.out

Enter⎵name⎵of⎵input⎵file:⎵**chicago-100.txt⮒**

$⎵

# Recommendations

1. Read the header file.
2. Use descriptive (long) naming conventions for variables and functions.
3. Add comments to the code to describe anything which is not obvious from the code.
4. Use whitespace (indentation, newlines) to visually organize code.
5. Use functions to reduce code duplication and increase abstraction.

# Getting Started

1. Start early.
2. Download the starter code.
3. Read the header file.
4. Compile and run the program.
   1. It won’t do anything, but it also won’t crash
5. Submit it to Mimir.
6. Pick a test and a function to implement first and implement it just enough to pass the test
   1. Plan your program on paper (digital or analog) before mashing the keyboard. Think about how your approach would handle the same input that we have in this prompt.
7. Recompile and rerun.
   1. Check for errors.
   2. If no errors, move on
   3. Else, start debugging
8. Resubmit to Mimir.
   1. If the target test is passing, move on
   2. Else, start debugging.  
      The input files used in Mimir are available in our Google Drive.  
      Consider running your local machine by creating a main function that invokes the function that you are working on. For example, if you are working on the mode function, your main function can be:
   3. #include <iostream>
   4. using std::cin, std::cout, std::endl;
   5. int main() {
   6. double array\_a[3] = {10, 10, 16};
   7. double array\_b[4] = {10, 16, 10, 16}:
   8. cout << "mode(array\_a, 3) returns "   
       << mode(array\_a, 3) << endl;
   9. cout << "mode(array\_b, 4) returns "  
       << mode(array\_b, 4) << endl;
   10. return 0;
   11. }
9. Continue by pickling a new test and writing just enough code to pass it (step 4)

# Submission

The source files to submit are named

1. main.cpp
2. taxi\_trips.h
3. taxi\_trips.cpp

# Have fun!

1. The mode is the most frequent value. [Read more on Wikipedia](https://en.wikipedia.org/wiki/Mode_(statistics)). [↑](#footnote-ref-0)