ARIZONA STATE UNIVERSITY

CSE 310 SLN 88400 — **Data Structures and Algorithms** — Fall 2018

Project #1

Available 08/29/2018; milestone due 09/12/2018; complete project due 10/01/2018

This project has two major goals:

1. The first is to implement various sorting and selection algorithms.
2. The second is to run experiments comparing the performance of certain algorithms, to collect and plot data, and to interpret the results.

Note: This project is to be completed **individually**. Your implementation **must** use C/C++ and ultimately your code **must** run on the Linux machine `general.asu.edu`.

All dynamic memory allocation **must** be done yourself, i.e., using either `malloc()` and `free()`, or `new()` and `delete()`. You may not use any external libraries to implement any part of this project. The use of string functions, e.g., `strcat`, `strcmp`, is allowed.

See §2 for the milestone and full project requirements. Scripts will be used to check the correctness of your program. Therefore, absolutely no changes to these project requirements are permitted.

You should use a version control system as you develop your solution to this project, e.g., GitHub. Your code repository must be *private* to prevent anyone from plagiarizing your work.

1 Major League Baseball

The World Series is the annual championship series of Major League Baseball in North America, contested since 1903 between the American League champion team and the National League champion team. A mountain of statistics is gathered for each Major League Baseball (MLB) game, player, player position, and team. In this project you will write a program that executes a sequence of commands that operate on MLB game statistics over a number of years.

All input data is to be read from standard input (`stdin`). Of course, you will redirect `stdin` from an input file. Similarly, your output must be written to `stdout`, which can be redirected to an output file. *Do not* use file operations to process files!

You may assume the format of the input data is correct.

The format of the input data is as follows:

- The number of years y (integer) of team game statistics in this data set.
- For year y the following is provided:
  - The total number of teams t in MLB for year y.
  - For each of the t teams the following statistics are provided in a tab separated list:
    - The team name (string),
    - the league, either American League (AL) or National League (NL) (string),
    - the number of games in which a player has appeared (integer),
    - the number of official at bats by a batter (integer),
    - the number of runs,
    - the number of times a batter hits the ball and reaches bases safely,
    - the number of times a batter hits the ball and reaches second base,
    - the number of times a batter hits the ball and reaches third base,
    - the number of home runs,

  - ∗ the number of runs that score safely,
  - ∗ the number of walks by a batter (four balls during an at bat),
  - ∗ the number of strikeouts by a batter,
  - ∗ the number of times a player has stolen a base,
  - ∗ the number of times a player has been put out attempting to steal a base,
  - ∗ the average number of hits by a batter defined by hits divided by at bats,
  - ∗ the on base percentage,
  - ∗ the slugging percentage, and
  - ∗ the on base percentage plus slugging.

A structure `struct mlb_stats` in §1.1, and in a file named `defns.h`, have been provided for your use.

Upon reading the value of y, you must dynamically allocate an array of `struct annual_stats` of size y, i.e., to hold statistics for each year. Then you initialize the array with the data for each year that follows.

Once the data is read and the structure holding the annual statistics structure is initialized, a sequence of commands to be processed follows. The input continues with:

- The number of commands c (integer).
- c commands following the format given in §1.2.

Each of the c commands must be processed as described in §1.2.

1.1 Format of Team Data

For each team, the structure `mlb_stats` contains numerous fields associated with each given team in a given year. The structure `annual_stats` holds team statistics for all teams in a given year. The data for this project is taken from the statistics pages for Teams on the Major League Baseball (MLB) site.

```
#define TEAM_NAME_LEN 50 // Maximum team name string length
#define LEAGUE_NAME 3 // Maximum league name string length,

struct mlb_stats{
   char Team[ TEAM_NAME_LEN ]; // Name of the MLB team
   char League[ LEAGUE_NAME ]; // American or National league, AL or NL
   int G; // Number of games
   int AB; // Number of official at bats by a batter
   int R; // The number of times a baserunner safely reaches home plate
   int H; // The number of hits
   int B2; // The number of times a batter hits the ball and reaches second base
   int B3; // The number of times a batter hits the ball and reaches third base
   int HR; // The number of home runs
   int RBI; // The number of runs that score safely
   int BB; // The number of walks by a batter
   int SO; // The number of strikeouts by a batter
   int SB; // The number of times a player has stolen a base
   int CS; // The number of times a player has been put out attempting to steal a base
   float AVG; // The average number of hits by a batter
   float OBP; // The on base percentage
   float SLG; // Slugging percentage
   float OPS; // The on base percentage plus slugging
};

struct annual_stats{
   int year;
   int no_teams; // Number of teams in the given year
   struct mlb_stats *stats; // Array size of statistics depends on number of teams in the given year
};
```

1.2 Commands

There are six (6) commands to implement. The syntax of valid commands is:

- `isort year field order`, use the Insertion sort algorithm to sort team data for the given `year` on the given `field` in the given `order`. The output is a list of team name, and field name, with appropriate headers on the list.
  - If the `year` given is not one of the years that is part of the input data, then output the error message: `Error: no such year.`
  - The `field` may be any field of `struct mlb_stats`. This structure contains fields of strings, integers, and floating point numbers. Hence the Insertion sort algorithm should be able to sort any of these types.
  - The `order` may be either `incr` for increasing order, and `decr` for decreasing order. If there are multiple fields with the same field value, then they should be sorted in alphabetic order by team name.
- `msort year field order`, use the Mergesort algorithm to sort team data for the given `year` on the given `field` in the given `order`. **Your Mergesort algorithm should allocate and deallocate a separate array for the Merge step, for each level of recursion except the base case.** See the command `isort` for a description of the `year`, `field`, and `order`, as well as the format of the output.
- `ifind year field select`, first uses Insertion sort to sort the team data for the given `year` on the given `field` in increasing order. Then, a selection is made based on `select`. Valid `select` for the selection include:
  - `max`, prints the maximum value for the given `field` in the given `year`, along with the team name achieving the maximum.
  - `min`, prints the minimum value for the given `field` in the given `year`, along with the team name achieving the minimum.
  - `average`, prints the average value for the given `field` in the given `year` over all teams. Only an integer or floating point `field` will be given for this `item`.
  - `median`, prints the median value for the given `field` in the given `year` over all teams. Here, "median" refers to the lower median, i.e., the element at position $\lfloor (n+1)/2 \rfloor$, where n is the total number of elements.

  In all cases, if there is more than one team satisfying the selection criteria, output all of them. Similarly, if there are no teams satisfying the selection criteria output the message to that effect.
- `mfind year field item`, first uses Mergesort to sort the team data for the given `year` on the given `field` in increasing order. See the command `ifind` for a description of the `year`, `field`, and `item`, as well as the format of the output.
- `isort range start-year end-year field order`, use the Insertion sort algorithm to sort team data for the range of years starting with `start-year` and ending with `end-year`, on the given `field` in the given `order`. There are two differences between this command and `isort` for a single year.
  - The sort is to be performed using data for all years in the given range. You may assume that `start-year` < `end-year` and that all years exist in the input data provided.
  - Add the year as part of the output for clarity.
- `msort range start-year end-year field order`, use the Mergesort algorithm to sort team data for the range of years starting with `start-year` and ending with `end-year`, on the given `field` in the given `order`. See the command `isort` on a range of years for a description of the `start-year`, `end-year`, `field`, and `order`, as well as the format of the output.

Consider the following valid example command sequence. Comments will not be part of the input, and are only included here for clarification.

```
6 // A total of 6 commands follow
isort 2018 RBI decr // Insertion sort 2018 data on RBI in decreasing order
msort 2018 HR decr // Quicksort 2015 data on home runs in decreasing order
ifind 2018 AB max // Return the team(s) with maximum AB after Insertion sorting 2018 data
ifind 2018 SLG average // Return team(s) with the average SLG after Insertion sorting 2018 data
mfind 2018 SO min // Return team(s) with fewest SO after Mergesorting 2018 data
msort range 2014 2018 BB incr // Mergesort 2014-2018 data on BB in increasing order
```

2 Program Requirements for Project #1

1. Write a C/C++ program that implements all of the commands described in §1.2 on data in the format described in §1.

2. Design experiments that exercise your program to answer the questions in §3. A brief report with figures and data to support your answers is expected.

3. Provide a `Makefile` that compiles your program into an executable named `p1`. This executable must run on `general.asu.edu`, compiled by a C/C++ compiler that is installed on that machine, reading input from `stdin` redirecting from a file in the prescribed format.

Sample input files that adhere to the format described in §1 will be provided on Blackboard; use them to test the correctness of your programs.

3 Experimentation

1. Plot the run time of your Insertion sort algorithm and your Mergesort algorithm on arrays of integers of size $n = \{10^1, 10^2, 10^3, 10^4, 10^5, 10^6\}$ (or larger if you wish). For this you will need to write a separate program that simply calls the Insertion sort and Mergesort algorithms you have implemented to meet the program requirements of Project #1, i.e., use your solution to the `isort` and `msort` commands for integers. The `clock` function or the `time` command may be used to time the execution of your program. A program to generate random integer data, named `randSeq.cc` will be provided.

   - Do you observe a cross-over point? That is, can you recommend an input size when you should use one algorithm over the other?

2. Plot the run time of maximum finding on arrays of integers of size $n = \{10^1, 10^2, 10^3, 10^4, 10^5, 10^6\}$ (or larger if you wish) using three techniques: Insertion sort the array in increasing order and then select the maximum, Merge sort the array in increasing order and then select the maximum. Specifically, you should be able to use your solutions to `ifind` and `ifind` on `item=max`, respectively, for this purpose. Use random integer data for this purpose.

   - Do you observe any cross-over points? Is there a point the recursive Mergesort algorithm overtakes the iterative Insertion sort?

4 Submission Instructions

1. The milestone deadline is before 11:59pm on Wednesday, 09/12/2018. See §4.1 for requirements.

2. The complete project deadline is before 11:59pm on Monday, 10/01/2018. See §4.2 for requirements.

It is your responsibility to submit your project well before the time deadline!!! Late projects are not accepted. Do not expect the clock on your machine to be synchronized with the one on Blackboard!

An unlimited number of submissions are allowed. The last submission will be graded.

4.1 Requirements for Milestone Deadline

By the milestone deadline, your project must implement the following commands as a minimum: `isort year`, `ifind year`, `isort range`.

You should also start collecting run times for the Insertion sort algorithm as described in §3.

Submit electronically, before 11:59pm on Wednesday, 09/12/2018 using the submission link on Blackboard for the Project #1 milestone, a zip[1] file named `yourFirstName-yourLastName.zip` containing the following:

Project State (5%): In a folder (directory) named `State` provide a brief report (.pdf preferred) that addresses the following:

1. Describe any problems encountered in your implementation for this project milestone.
2. Describe any known bugs and/or incomplete command implementation for the project milestone.
3. While this project is to be complete individually, describe any significant interactions with anyone (peers or otherwise) that may have occurred.
4. Cite any external books, and/or websites used or referenced.

Implementation (50%): In a folder (directory) named `Code` provide:

1. In one or more files, your well documented C/C++ source code implementing the commands required for this project milestone.
2. A `Makefile` that compiles your program to an executable named `p1` on `general.asu.edu`. Our TA will write a script to compile and run all student submissions on `general.asu.edu`; therefore executing the command `make p1` in the `Code` directory must produce the executable `p1` also located in the `Code` directory.

Correctness (45%): The correctness of your program will be determined by running a series of commands on a variety of MLB team data, some of which will be provided to you on Blackboard prior to the deadline for testing purposes. For the milestone deadline, the scripts will only contain the subset of the commands described in §1.2 and listed above. As described in §2, your program must read input from standard input. **Do not use file operations to read the input!**

The milestone is worth 30% of the total project grade.

4.2 Requirements for Complete Project Deadline

Submit electronically, before 11:59pm on Monday, 10/01/2018 using the submission link on Blackboard for the complete Project #1, a zip[2] file named `yourFirstName-yourLastName.zip` containing the following:

Project State (10%): Follow the same instructions for Project State as in §4.1.

Experimentation and Report (10%): In a folder (directory) named `Report` provide a brief report (.pdf preferred) that addresses the following:

1. Describe the experiments you ran, i.e., the machine used to run the experiments along with some of its characteristics, the input data sizes, and so on.

[1] **Do not** use any other archiving program except `zip`.
[2] **Do not** use any other archiving program except `zip`.

2. Present figures plotting the results of your experimentation as requested in §3. Use the data you collected to answer to the questions found in the same section.

Implementation (40%): Follow the same instructions for Implementation as in §4.1.

Correctness (40%): The same instructions for Correctness as in §4.1 apply except that the input files will exercise all commands from §1.2 rather than a subset of them.

Your zip file for the full project submission must contain the contents of 3 folders: `State, Report,` and `Code`.

5 Marking Guide

The project milestone is out of 100 marks.

Project State (5%): Summary of project state, use of a zip file, and directory structure required (i.e., a folder/directory named `State` and `Code` is provided).

Implementation (50%): 40% for the quality of implementation in your code including proper memory management for the input, 10% for a correct `Makefile`.

Correctness (45%): 40% for correct output on several files of sample input, 5% for redirection from standard input.

The full project is out of 100 marks.

Project State (5%): Summary of project state, use of a zip file, and directory structure required (i.e., a folder/directory named `State`, `Report`, and `Code` is provided).

Experimentation and Report (10%): Experiment design, results of experimentation, and answers to questions.

Implementation (40%): 30% for the quality of implementation in your code including proper memory management, 10% for a correct `Makefile`.

Correctness (45%): 40% for correct output on several files of sample input, 5% for redirection from standard input.