CSCI-1200 Data Structures — Fall 2015
Homework 2 — League of Legends Classes

In this assignment you will parse, compute, and study statistics from an obscure video-based endeavor called "League of Legends". As it is unlikely that you know anyone who has played this sport, we will summarize all of the rules that you need to know to complete this homework. *We certainly do not recommend you engage in such activities before you have completed your homework!*

A head-to-head *match* in League of Legends consists of two *teams* of 5 *players*. At the start of the match, each player selects a *champion* from a list of (currently) 126 champions, each with their own abilities. Players may coordinate to select champions with complementary strengths to improve the performance of their team in the match. During the match, each player controls his or her champion character and battles with players on the opposite team and also computer-controlled *minion* characters. When combat results in the *death* of a champion, that player must wait a small period of time before their champion *re-spawns* and they can re-join the match. Credit for the *kill* of a champion character is given to the appropriate player on the other team, or in some cases to a general game minion. One or more of the teammates may also be given credit for *assisting*. The game ends when one team destroys the other team's *Nexus*.

More details on the game are available here: https://en.wikipedia.org/wiki/League_of_Legends. Data for this homework comes from actual gameplay, made publicly available by the game author, *Riot Games, Inc.*: https://developer.riotgames.com/docs/getting-started. We have heavily reduced these very large and detailed datasets to a minimum set of information.

Here is an example of the input format for this homework (some event lines are omitted for space):

```
MATCH ID 1778973240
LOSING TEAM
  iTcnRKaneki playing champion Syndra
  T1Zed playing champion Kalista
  IamZevahc playing champion Thresh
  SushiNChopstix playing champion Riven
  BattlecastLulu playing champion Udyr
WINNING TEAM
  Dopy playing champion Graves
  komimatt playing champion Brand
  SRVaRiaX playing champion Lissandra
  Diddydan playing champion Zac
  Jrazo playing champion Blitzcrank
EVENTS
  @ 277230 BattlecastLulu [ SushiNChopstix ] killed SRVaRiaX
  @ 432589 Dopy [ Jrazo ] killed T1Zed
  @ 444272 SushiNChopstix [ T1Zed IamZevahc ] killed Dopy
  @ 448804 minion killed SushiNChopstix
  @ 519016 T1Zed [ IamZevahc ] killed Dopy
           . . .
  @ 2410278 Dopy [ komimatt SRVaRiaX Jrazo ] killed BattlecastLulu
END
```

First we see the `MATCH ID`, then the players and champions for each team are listed. The teams are labeled "`WINNING`" and "`LOSING`", indicating the final result of the match. The highlights of all *champion kill* events are listed chronologically. Each of these events begins with the '`@`' character, then a timestamp (in milliseconds from the start of the match), the name of the player given primary credit for the kill, the assistants (if any) listed in square brackets, and finally the name of the victim. At the end of the events list is the keyword "`END`". An input file will contain one or more matches.

File I/O and Command Line Arguments

Your program will expect 3 required command line arguments. The first and second are the names of the input and output files, respectively. The third argument is the name of the output table that should be printed, one of "`players`", "`champions`", or "`custom`". For example, here are some valid command lines to your program:

```
./lol_stats.out input_simple.txt  output_simple_players.txt   players
./lol_stats.out input_small.txt   output_small_champions.txt  champions
./lol_stats.out input_medium.txt  output_medium_custom.txt    custom
```

Statistics Collected and Output

The output will be one of three different tables, as specified on the command line.

First, we study the "`players`" table, containing all players who participated in one or matches in the input file. Each row contains the number of kills credited to this player, the number of times they died and respawned during the match, the *kill-to-death ratio* (KDR), and the names of the champion(s) used by that player. *NOTE: To avoid divide by zero, if a player's deaths is zero, then the KDR is equal to the kills.* If a player played more than one champion in different matches, the names of the champions should be sorted alphabetically, separated by commas (without duplicates). Overall, the rows of this table are sorted by KDR. If two players are tied in KDR, they are sorted by kills (highest first), then by deaths, and if tied in all of the above, they should be alphabetically by player name. *Note: We will simply use the less than $<$ operator on STL strings, which places numbers before capital letters, and capital letters before lowercase letters.*

Here is an example of this output:

PLAYER NAME	KILLS	DEATHS	KDR	PLAYED WITH CHAMPION(S)
squirTOT	6	1	6.00	Amumu
komimatt	7	2	3.50	Brand
KelloggsBRNFLKZ	10	3	3.33	Sejuani
Diddydan	3	1	3.00	Zac
AntiToro	13	6	2.17	Tristana
Dopy	10	6	1.67	Graves
Jrazo	4	3	1.33	Blitzcrank
alittlemido	5	4	1.25	Thresh
OMGLeiChen	9	10	0.90	Graves
Jovone	7	8	0.88	Nasus
SRVaRiaX	6	7	0.86	Lissandra
iTcnRKaneki	6	7	0.86	Syndra
Stylether	10	12	0.83	LeBlanc
SushiNChopstix	6	8	0.75	Riven
T1Zed	6	8	0.75	Kalista
Indovaasion	5	10	0.50	Fizz
Zraotic	4	9	0.44	Warwick
BattlecastLulu	1	3	0.33	Udyr
IamZevahc	0	5	0.00	Thresh
Zyyke	0	7	0.00	Taric

Next, we study the "`champions`" table, which contains all champions that were used in one or more matches in the input file. This table contains the number of wins & losses, the corresponding win percentage, and the number of times that champion was killed not by another champion, but by a minion (rather embarrassing if you think about it...). Overall this table is sorted by win percentage, highest first. If tied in win percentage, champions should be sorted by wins, then by losses, and finally by name if all of the above are equal.

And here is an example of this type of table:

CHAMPION NAME	WINS	LOSSES	WIN%	MINION DEATHS
Amumu	1	0	1.00	0
Blitzcrank	1	0	1.00	0
Brand	1	0	1.00	0
LeBlanc	1	0	1.00	1
Lissandra	1	0	1.00	0
Sejuani	1	0	1.00	0
Tristana	1	0	1.00	0
Zac	1	0	1.00	0
Graves	1	1	0.50	0
Thresh	1	1	0.50	0
Fizz	0	1	0.00	0
Kalista	0	1	0.00	0
Nasus	0	1	0.00	0
Riven	0	1	0.00	1
Syndra	0	1	0.00	0
Taric	0	1	0.00	0
Udyr	0	1	0.00	0
Warwick	0	1	0.00	0

Finally, the "`custom`" table is a chance for you to be creative. You will collect, organize, and output some other statistic from the provided match data. Notice that the earlier tables didn't use the event timestamp, or the champion kill assists. You may be interested in exploring that data. Can you identify champions who are good *supporting* team members (many assists, but not many kills)? Can you identify the best *carry* players, who are strong late in the match and help carry the team to victory? Or perhaps you can study pairs of champions who are more successful when they play on a team together. Or find champions who are particularly susceptible to a team with a specific enemy champion.

The most important task for this part of the assignment is to write a concise description ($\sim$ 100 words) of your new statistic. What is your hypothesis for the patterns that might emerge from your statistic? Put this description in your `README.txt` along with any other notes for the grader. Be sure to tell the grader which dataset best demonstrates your new statistic. You may also create your own dataset and include it and your program's output for that test case with your submission. Extra credit will be awarded to particularly interesting statistics that require clever programming.

Useful Code

To control the formatting of your tables, you'll want to read up on the various iomanipulators: `std::setw(int)`, `std::setprecision(int)`, `std::fixed`, `std::left`, and `std::right`. And don't forget about the `sort` function that can be used to order the contents of a `vector`.

We encourage you to use the `<<` input stream function for strings to implement all of the parsing necessary for this assignment. Be sure to study the more complex parsing example on this website: Misc. C++ Programming Information. *Note: You do not need to use* `getline` *or* `eof`*. In fact, your code should not rely on the position of newlines or whitespace within the input file.*

Program Requirements & Submission Details

Your program should involve the definition of *at least one well-designed C++ class* that has its own `.h` and `.cpp` files, named appropriately.

Be sure to read the "Homework Grading Criteria" as you put the finishing touches on your solution. Use the provided template `README.txt` file for notes you want the grader to read. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.