ENGG1003
Introduction to Procedural Programming
Programming Assignment 2 – v.2
Due: Friday, June 1st 2018 @ 23:59pm

Submission is by a single (zipped) Visual Studio Solution folder to Blackboard via Assessments tab. (Course Value is 12.5%)

PREPARATION:

Preparation for this assignment mostly consists of making sure that your Visual Studio 2015 Solution and Project Spaces for the assignment are set up correctly so that you can easily submit the required files. You will need to set up a new solution called ***cstudentnumber*PA2** within your **U:\GENG1003** folder. Within the solution have a single project called **Assignment2**.

PROBLEM STATEMENT:

The Soccer World Cup is one of the main sporting events in the world. The game is called Soccer in Australia because we play 4 different types of football; however, most of the world simply refers to it as Football. The International Federation of Associated Football – FIFA (Federation Internationale de Football Association) runs this competition every four years, and this year it will be in Russia.

Unfortunately, FIFA has lost all programs that report countries' positions and scores. Currently, a company has been hired to develop a completely new application, but it won't be ready until the knockout stage of the competition. Therefore, there is a need to report on the groups and team positions including everyday results, so news can be reported. Thus, the program needs to be re-written in its simplest possible form to supply this information. The program will just use simple text files for input.

You will need to write a program that allows FIFA to feed in simple text files of group teams and soccer matches results and produce lists that are ordered per group, alphabetically, as well as, in country performance order.

BACKGROUND:

The Soccer World Cup features 32 national teams competing over the course of a month in the host nation. There are two stages: the group stage followed by the knockout stage. In the group stage, teams compete within eight groups of four teams each. Each group plays a round-robin tournament, in which each team is scheduled for three matches against other teams in the same group. This means that a total of six matches are played within a group. The top two teams from each group advance to the knockout stage. Points are used to rank the teams within a group. Three points are awarded for a win, one for a draw and none for a loss.

The soccer national teams within the group are ranked, initially, according to this given points; however, in some cases there could be ties. When there is a tied position, this one is solved as per the following criteria:

1. Team with the greatest goal difference in all group matches gets the position.
2. Team with the greatest combined number of goals scored in all group matches gets the position.
3. Team with the greatest number of points in head-to-head matches among those tied teams gets the position.
4. Team with the greatest goal difference in head-to-head matches among those tied teams gets the position.
5. Team with the greatest number goals scored in head-to-head matches among those tied teams gets the position.
6. If more than one team remain level after applying the above criteria, their ranking will be determined by the drawing of lots.

For the purposes of this simple program and since it is just for the robin-round tournament, the performance ranking is done by the points criteria, and in case of a tie, criteria 1 and 2 above. If the tie still persists, alphabetical order will be norm. NOTE: You don't need to worry about criteria 3 to 6, and in case you have already implemented it with criteria 3 to 6, it is ok to use them.

The knockout stage is a single-elimination tournament in which teams play each other in one-off matches, with extra time and penalty shootouts used to decide the winner if necessary. It begins with the round of 16 (or the second round) in which the winner of each group plays against the runner-up of another group. This is followed by the quarterfinals, the semi-finals, the third-place match (contested by the losing semi-finalists), and the final.

In order to properly report the results needed, you will need to hold data for each country comprising: Team Name (a character array large enough to hold the longest name – say 16 characters), Group, Games Played, Games Won, Games Drawn, Games Lost, Goals For, Goals Against, Points. The above data is best stored in a C-struct for each team. If you want, you can also create another C-struct for groups, but definitely, it will be best to work with an array of these struct(s) being the complete data held for the World Cup.

For the 32 Soccer World Cup teams, we will use shortened team names, so there won't be any space between words. Results for a World Cup game are reported giving the names of the teams and the number of goals scored, e.g. TeamA 3 TeamB 2. Note that this means that TeamA has won the game 3-2, while TeamB has lost the game 2-3. If the number of goals for each team is equal, then the game result is a draw, e.g. TeamC 1 TeamD 1 would be a 1-1 draw for both teams.

TASKS:

Your program will read two data files, set up in the same folder as was done for the first assignment. These data files are called *team.dat* and *match.dat*.

You program should have the following steps:

1. Read the national football team names and the group they belong to from the first data file and store them (wherever your program design says it is best to do so). There will be 32 names, and each group contain 4 teams. You may then close the first file. You may assume

that this first data file does not contain any errors at all. The file contains the team names ordered per group.

2. Read the second data file, reading and processing one data set at a time, stopping at end-of-file. The second data file will contain data for all the games played during the current time period. Each data set in this file will consist of a result for an individual game, and will consist of 2 team names, each followed by a number of goals (that is, each data set will have the form: *first_team first_score second_team second_score*). It will also be possible for this file to be empty.
3. For each data set, check for data consistency. If any data item is negative the data set is invalid. Any invalid data set is reported with an error message and ignored.
   E.g. ***Colombia 3 Poland -1 – data is invalid***
4. For each data set, find the correct team's name in your data structure. If the team's name cannot be found, including matching exactly by case, then the data is also invalid, whereupon it is reported as such, and then ignored.
   E.g. ***Columbia 2 Japan 1 – name not found in team list***
   australia 1 France 1 – name not found in team list
5. Once the data set is valid, for each match, you will need to record the result for each team (that is the win for one team and the loss for the other, or a draw for both of them), and the goals counters. This will involve updating the data values for the respective teams. You will need to search for the names of the teams and record the respective result for each team.
6. Produce a by performance by group decreasing ordered list of the teams as per required table of reporting (below). Use an output format that will left-justify the name. This should be reasonably simple since the team names were originally input in by group order.
7. Produce (sort) an alphabetically ordered list of the teams and their performances as per required table of reporting (below). Use an output format that will left-justify the name.
8. Produce (sort) a list of teams by decreasing order of team performance as per required table of reporting (below). This will be a challenging task – think carefully about how you will compare the ranking of any two teams – this becomes the basis for sorting.

Observation: While it is possible to complete this assignment using a number of parallel arrays, your design will be better and easier to work with, if you define your own structure(s) type (it is best to use a naming convention of *structname_t*), and then have a single array of these objects. Your design will also benefit from having a number of functions. A solution that does not have well designed functions to perform major tasks will be marked down significantly.

OUTPUT:

Use output formats that will allow normal data to be output in columns. Use headings as appropriate to display the data as a table along the following guidelines (the actual numbers used here are fictitious):

```
FIFA World Cup – Russia 2018
Program Developed by Cesar Sanin
Student Number c12345678
Computer Lab Monday 1 pm - 4 pm

Opening files.
Processing…
      Colombia 3 Poland -1  – data is invalid
      Columbia 2 Japan 1 – name not found in team list
      australia 1 France 1 – name not found in team list
Valid data sets 45
Closing files.
```

Name should be left-justified as stated above, all numerical columns are a maximum of 3 digit integers, but have more characters wide (as per the column label plus spaces). E.g.

```
Team Results  -  Group Order

-------------------------------------------------------------------------
Group  Name          Points  Played   Won  Drawn  Lost   For   Against  Goal-Diff
-------------------------------------------------------------------------
   A   Uruguay          7       3       2     1      0     7      2          5
       Russia           7       3       2     1      0     4      1          3
       Egypt            3       3       1     0      2     6      6          0
       SaudiArabia      0       3       0     0      3     1      9         -8
   B
   …   …                …       …       …     …      …     …      …          …
```

Note that the goal difference is simply Goals Scored (For) minus Goals Conceded (Against).

The second list output will look similarly to the following table, e.g.

```
Team Results  -  Alphabetical Order

-------------------------------------------------------------------------
Name         Group  Points  Played   Won  Drawn  Lost   For   Against  Goal-Diff
-------------------------------------------------------------------------
Argentina      D      7       3       2     1      0     7      2          5
Australia      G      7       3       2     1      0     4      1          3
Belgium        C      3       3       1     0      2     6      6          0
Brazil         E      0       3       0     0      3     1      9         -8
     …      …          …       …       …     …      …     …      …          …
```

The third list output will look similarly to the following table, e.g.

```
Team Results  -  Performance Order

-------------------------------------------------------------------------
Name         Group  Points  Played   Won  Drawn  Lost   For   Against  Goal-Diff
-------------------------------------------------------------------------
Colombia       H      9       3       3     0      0     7      2          5
Australia      C      6       3       2     0      1     4      1          3
Brazil         E      3       3       1     0      2     6      6          0
Spain          B      0       3       0     0      3     1      9         -8
     …      …          …       …       …     …      …     …      …          …
```

DATASETS:

Input file *team.dat* is provided in Blackboard. It includes the 32 football teams organized by their groups. Note that none of the team names contain spaces (this will make string input easier). These will appear correctly in the first data file (*team.dat*).

The only errors possible in the first input file will be as follows:

1. The File is non-existent.

Input file *match*.dat is provided in Blackboard with some data samples. Your program will be graded on a number of data sets for the second data file. While the file containing the team list will always have 32 names, the second data file may have any number of complete data sets.

If you want to use proper data, you can look at the last World Cup Brazil 2014 results (https://en.wikipedia.org/wiki/2014_FIFA_World_Cup#Group_stage). Otherwise, if you follow a particular team, you can dream about that team doing well by making up your own data.

It will be normal for these data to be one set of data (4 values) per line. As games are played, extra data can then be easily added to the end of the *match.dat* file.

The only errors possible in the second input file will be as follows:

2. The File is non-existent or empty.
3. A team name may be misspelt (i.e. different to the name given in the first data file). This will result in your array/struct search not finding one of the team names given in the data set.
4. The number of goals scored by a team may be given as negative.

In each case, you print out the data values read in with an appropriate warning message, and ignore the data values for that game (that is you don't update any team, even if one of the team names is correct), and you continue on to the next data set.

OTHER PROGRAMMING REQUIREMENTS

- No global variables are allowed (struct definitions are not variables).
- Your program needs to display all information such that is easy to read and as per the sample output presented.
- Your code also must be very well commented and have a head of code for the programs and for the functions.
- You should use all methods you create. You can create any functions/methods that you considered necessary.

Sorting is also commonly used in computing. You may sort the array using the ***Bubble Sort*** algorithm, shown here:

```
void BubbleSort (int A[], const int arraySize) {
      int i, j;
      for(i = 0; i < arraySize; i++)
            for(j = 0; j < arraySize-1; j++)
                  if(abs(A[j]) > abs(A[j+1]))
                        swap(A+j, A+j+1);
}
```

Note that this algorithm sorts the array **A** in place. Note also that it sorts by the absolute values of the elements of **A**. Finally, note that it uses array arithmetic to pass pointers to the swap function.

The **swap** function is

```
void swap (int *a, int *b) {
      int temp = *a;
      *a = *b;
      *b = temp;
}
```

INPUT FILE SPECIFICATION:

To assist in the Grading of your assignment, you are required to submit your assignment with the input file names within the file open statement set to the strings **"team.dat"** and **"match.dat"**. This means that if you work on your assignment at home you will need to modify it and re-test it at University, and change any file name **BEFORE** you submit it. It also means that when you test your program at University your input files should be located in the same folder where you set up your Solution folder when you first began the assignment.

INSTRUCTIONS FOR SUBMISSION – PLEASE READ THIS CAREFULLY!!

Preparation for submission (similar to PA1):

1. Go to your **U:\GENG1003** folder where the solution was set up.
2. Right-click and hold on the **c*studentnumber*PA2** folder and choose Send To –> Compressed Folder.
3. This will create a file called **c*studentnumber*PA2.zip**. You will now need to add in the Assessment Item Cover Sheet.
4. Click and hold on the zip file and choose Open with -> Windows Explorer.
5. Then drag and drop your completed Assessment Item Cover Sheet file onto this window so that it is displayed alongside the solutions directory name (do not drag the cover sheet file inside the solutions folder name). This will add the Assignment Cover Sheet to the zip file.
6. Close the Windows Explorer window.

Submit your updated zip file via the link for PA2 in the Assessment tab on Blackboard.

EXTRA NOTES:

Please remember that incorrectly submitted assignments will not be marked**.**

- **Remember, a completed Assessment Cover Sheet must accompany your submission.** A copy of it will be placed on Blackboard, in the Assignments section. Assignments without a completed Assessment Item Cover Sheet **will not be further marked.**
- Assignments should contain the Visual Studio project, compressed into a .zip file. Only one file per student will be allowed.
- Assignments containing a compressed Visual Studio solution or project that was created outside the correct specification (see section PREPARATION) **will be discounted according to the amount of time wasted by the marker in correcting the submission – if the submission cannot be made "runnable" within a reasonable amount of time then it will not be further marked.**
- Your program file should adhere closely to the layout and commenting standards presented in class, and your code should be properly documented with inline comments.

Remember that this assignment will take many hours to complete, and the submission itself should take less than 10 minutes. **Do not waste your effort by submitting it incorrectly.**

Late submissions are subject to the rules specified in the Course Outline.

This assignment is worth 12.5% of your final result for the course.

Sanin - 2018