COMP 1633 Assignment 2 ("Olympic Hockey Teams List" Manager)

Given: Monday, January 31, 2022
Electronic Copy Due: Friday, February 18, 2022 at 11:59 p.m.
Hard Copy Due: not required

Note: in first year, linked list assignments are always harder than they first seem. The algorithm design and debugging phases always require more time than students anticipate. *Start early!* It is very important that you work in an incremental fashion and to encourage this you will be required to use Google Test to test some of the list processing functions required for this assignment.

These specifications are long and detailed. It is important to read them carefully and follow them explicitly.
The marking of the program output is being done by computer so your output must match exactly – if it does not, this will result in a grade deduction. The exact format of error messages is shown later in the specs and an executable version of the program is being provided so the program output can be observed and copied.

Objectives

- To build problem solving skills.
- To build C++ programming skills.
- To learn about singly linked lists, including building and traversing them.
- To develop linked list algorithms for node insertion and removal.
- To apply the insertion sort algorithm to linked lists.
- To gain experience with pointers.
- To gain experience with dynamic memory allocation.
- To gain experience testing and debugging more difficult algorithms.
- To gain experience using an external library.
- To gain experience using Google Test to test a library module
- To gain experience writing good documentation.

The Problem

As the opening of the XXIV Winter Olympic Games in Beijing, draws closer, Canadians nationwide are hoping that the Women's Ice Hockey Team can regain the gold medal. Historically, Canada has dominated the event since the sport of women's hockey was introduced in 1998. In the six Olympic games since 1998 Canada has won the gold medal four times and silver twice. The defending medalists from the 2018 PyeongChang games are USA, Canada and Finland.

In the XXIV Games, Canada, the USA, Finland and Olympic Athletes from Russia are expected be the primary contenders. The six other nations (Switzerland, Japan, the Czech Republic, Sweden, Denmark and China) are considered on the fringe and their chances of winning medals are low.

Avid fans will be delighted to have their own way of keeping track of the standings and each player's points total. So, we're going to build them a little application to do this. For simplicity, we'll only use 3 teams – Canada, USA and Finland – although once the basic structure is in place, it would be easy to expand to maintain records for all the teams.

The Program

Write a program to help a fan build and manage an "Olympic Teams" list for women's ice hockey.

When the program is run it prompts the user to enter the name of a text file; for example, `team_rosters.txt`. If the user enters an empty string, the program starts with an empty "Olympic Teams" list. Otherwise, it attempts to open the file and, if successful, creates an "Olympic Teams" list in memory of the players in the file. If the file cannot be opened successfully the program should gracefully exit, i.e. print an appropriate message and terminate normally.

To keep the player entries simple, the program forces them to have a fixed format. For example, `team_rosters.txt` might look like this:

```
C f 9 6 0 Sarah Nurse
C f 7 2 0 Marie-Philip Poulin
C d 0 3 0 Jocelyne Larocque
C g 0 1 3 Emerance Maschmeyer
F f 1 4 0 Noora Tulus
F f 7 5 0 Susanna Tapani
F g 0 0 5 Anni Keisala
U f 3 3 0 Amanda Kessel
U g 0 0 5 Nicole Hensley
U d 0 1 0 Megan Keller
I d 0 2 0 Jenni Hiirikoski
I f 2 4 0 Brianna Decker
I g 0 0 4 Kristen Campbell
```

The character at the beginning of a line is the player's team (C = Team Canada, F= Team Finland and U = Team USA). All of Canada's players come first, followed by Finland's players, and then the USA's players. The team designated '`I`' is the Injury Reserve list. Players from the Injury Reserve list may be moved to any designated team, and players from any team may be moved to this list.

The second character is the player's position: `f` for forward, `d` for defence and `g` for goaltender.

The next three integers store data about the player's accumulated points:

- the first integer is the player's accumulated goals
- the second integer is the player's accumulated assists
- the third integer is the player's goals against.

When the player's position is "forward" or "defence", the goals against will be zero. When the player's position is "goaltender", the first two integers will usually (but not always) be zero while the third entry indicates the number of goals allowed by the goalie. No error checking for these is required.

The rest of the line is the player's name, which must be between 1 and 25 characters in length, including any spaces.

Because a player file is a text file, it could be edited which might introduced errors. However, for the purposes of this assignment, we will assume that this never happens. Therefore, the program will not be required to check for any errors in the input file data, such as a player on Team USA appearing before a player on Team Canada. You should note, however, zero players on a particular team is not an error; in fact, the entire player file could be empty.

When the program exits, the user is prompted for the name of an output file. If the file entered exists (for example, the file that the list was originally read from), then it will be overwritten. Otherwise, a new file is created.

The program then traverses the list and writes the player entries to the file in exactly the format shown above. After writing the list to the file the nodes in the list are to be deallocated and the pointer to the head of the list is set to NULL. (Note 1: this is done via a function that is described later in the specs. Note 2: Strictly speaking memory deallocation is not required in this program, because the program is terminating. However, it is good programming practice to deallocate any dynamically allocated memory – so you will follow this practice. Marks are allocated for correct list management, which includes appropriate deallocation of dynamic memory, so this will be checked.)

Once the initial "Olympic Teams" list has been read, the program will repeatedly display the following main menu and then process the chosen command:

```
Olympic Teams List Manager:

d) display list
m) move a player to injury reserve
a) activate a player from injury reserve
u) update player points
s) show top scorer

q) quit

Your choice:
```

Command 'd': display list

The "display" command will display the current list using the following format. You do not have to match the exact number of spaces shown below, but spaces and blank lines must be in the locations shown below :

```
Team Canada:

1:      Sarah Nurse               9  goals    6  assists
2:      Marie-Philip Poulin       7  goals    2  assists
3:      Jocelyne Larocque         0  goals    3  assists
4:      Emerance Maschmeyer       0  goals    1  assists      3  goals against

Team Finland:

5:      Noora Tulus               1  goals    4  assists
6:      Susanna Tapani            7  goals    5  assists
7:      Anni Keisala              0  goals    0  assists      5  goals against

Team USA:

8:      Amanda Kessel             4  goals    3  assists
9:      Nicole Hensley            0  goals    0  assists      5  goals against
10:     Megan Keller              0  goals    1  assists

Injury Reserve:

11:     Jenni Hiirikoski          0  goals    2  assists
12:     Brianna Decker            2  goals    4  assists
13:     Kristen Campbell          0  goals    0  assists      4  goals against
```

Note that the player entries are numbered in the order in which they are displayed. These numbers will play a part in the "move to injury reserve", "activate from injury reserve" and "update" commands

The next three commands, move (‘m’), activate (‘a’) and update (‘u’), all require additional user input. In all cases the first two inputs are identical. The first input required is the player number which is the current list position for the player. The list position must be a value greater than zero. However, if the value is larger than the number of players on all lists an appropriate error message should be output and the user will be forced to enter a valid list position. If the list position is valid then a message stating the action to be taken and the player’s actual name will be output and the user will be allowed to enter either ‘n’ or ‘y’. If the user enters ‘n’ then the action is terminated with no change to the list; otherwise the requested action is completed. In the cases of activate and update additional input is required and will be discussed below.

Command 'm': move a player to injury reserve

A player can be moved from a team list onto the injury reserve list. If an active player is selected to be moved they are added to the end of the Injury Reserve list. Attempting to move a player that is already on the injury list onto the injury list is not allowed. If this occurs an appropriate error message should be output and the command should terminate with no action being taken. The "move" command behaves as follows:

```
Enter player number: 2

Move Marie-Philip Poulin to Injury Reserve (y/n)? y
```

The display command would now show:

```
Team Canada:

1:      Sarah Nurse               9  goals    6  assists
2:      Jocelyne Larocque         0  goals    3  assists
3:      Emerance Maschmeyer       0  goals    1  assists      3  goals against

Team Finland:

4:      Noora Tulus               1  goals    4  assists
5:      Susanna Tapani            7  goals    5   assists
6:      Anni Keisala              0  goals    0  assists      5  goals against

Team USA:

7:      Amanda Kessel             4  goals    3  assists
8:      Nicole Hensley            0  goals    0  assists      5  goals against
9:      Megan Keller              0  goals    1  assists

Injury Reserve:

10:     Jenni Hiirikoski          0  goals    2  assists
11:     Brianna Decker            2  goals    4  assists
12:     Kristen Campbell          0  goals    0  assists      4  goals against
13:     Marie-Philip Poulin       7  goals    2  assists
```

Command 'a': activate a player from injury reserve

The "activate from injury reserve" command behaves as follows:

```
Enter player number: 11

Activate Jenni Hiirikoski (y/n)? y
Enter team (C/F/U): F
```

The user will be forced to enter a valid team character. The activated player must be added at the end of the sub-list of the given team. Attempting to activate a player that is not on the injury list is not allowed and should result in an appropriate error message and the command terminating without any action being performed. For example, if the display command was now re-executed it would display:

```
Team Canada:

1:      Sarah Nurse               9  goals     6  assists
2:      Jocelyne Larocque         0  goals     3  assists
3:      Emerance Maschmeyer       0  goals     1  assists      3  goals against

Team Finland:

4:      Noora Tulus               1  goals     4  assists
5:      Susanna Tapani            7  goals     5   assists
6:      Anni Keisala              0  goals     0  assists      5  goals against
7:      Jenni Hiirikoski          0  goals     2  assists

Team USA:

8:      Amanda Kessel             4  goals     3  assists
9:      Nicole Hensley            0  goals     0  assists      5  goals against
10:     Megan Keller              0  goals     1  assists

Injury Reserve:

11:     Brianna Decker            2  goals     4  assists
12:     Kristen Campbell          0  goals     0  assists      4  goals against
13:     Marie-Philip Poulin       7  goals     2  assists
```

Command 'u': update player points

The "update" command allows the user to change the number of points for an active (i.e. not on the Injury Reserve list) player. For example, the game results for the day could be used to do daily updates to the Olympic teams' points. The user could enter:

```
Enter player number: 8

Update points for Amanda Kessel (y/n)? y

Enter goals to add: 1
Enter assists to add: 0
```

The display command would now show:

```
Team Canada:

1:      Sarah Nurse               9  goals     6  assists
2:      Jocelyne Larocque         0  goals     3  assists
3:      Emerance Maschmeyer       0  goals     1  assists      3  goals against

Team Finland:

4:      Noora Tulus               1  goals     4  assists
5:      Susanna Tapani            7  goals     5   assists
6:      Anni Keisala              0  goals     0  assists      5  goals against
7:      Jenni Hiirikoski          0  goals     2  assists

Team USA:

8:      Nicole Hensley            0  goals     0  assists      5  goals against
9:      Megan Keller              0  goals     1  assists
10:     Amanda Kessel             5  goals     3  assists

Injury Reserve:

11:     Brianna Decker            2  goals     4  assists
12:     Kristen Campbell          0  goals     0  assists      4  goals against
13:     Marie-Philip Poulin       7  goals     2  assists
```

Command 's': show Top Scorer

The "show Top Scorer" command displays the top scoring non-goaltender from among the active players. This is based on total points, i.e. the sum goals and assists. If there are no active players on the list or if there are only goaltenders on the list then a different appropriate message should be output to the user. For the purpose of this assignment, you may assume that a tie for the top scorer will not occur. For example

```
Top Scorer is:

Sarah Nurse with 15 total points
```

Command 'q': quit

The "quit" command will write the "Olympic Teams" list to a text file specified by the user. It will behave as follows:

```
Enter name of output file: newplayerlist.txt

Player list saved to file and program exiting.
```

I/O and Error Handling

The ioutil module discussed in the IOstream errors lecture is provided (via `ioutil.o` and `ioutil.h`) and must be used for all user input. You will need to review the documentation in `ioutil.h` so that your program correctly uses the provided functions. Failure to use these functions – or using them incorrectly - will result in a grade deduction. The data used to test your program will include test cases to ensure that this module has been used.

Should the user type `ctrl-d` for any input, this is to be treated as invalid input and the user forced to enter valid input.

The program must handle the following errors, the wording and formatting shown below must be used exactly!:

Error	Action
couldn't open input file	display `**** ERROR opening input file - program terminating ****` and exit gracefully
couldn't open output file	display `**** Unable to open output file – exiting ****` and exit program without saving the list to a file
User input errors: Your program needs to handle all five of the following errors. The first four are type `I` errors (see the IOstream errors notes) which the client program must handle. eof is caught by ioutil and indicated to the calling program via the eof flag. All other possible input errors will be handled by the ioutil operations!	The corresponding error message should be displayed **exactly as shown**. For all non-eof input, excluding an eof as a menu option, the user is to be forced to enter a valid value. Invalid menu selections will be handled by the menu processing code. The other four should be handled in the appropriate command handler code.
• invalid menu option selected (including eof)	display `**** ERROR - Invalid menu choice - try again ****`
• invalid team character	display `**** Team must be one of C(anada), F(inland) or U(SA) ****`

- a list item number <= 0
  where max in the error message is the actual maximum value

  display
  `**** Player number must be between 1 and max ****`

- a negative goal or assist value

  display
  `**** All player points must be greater than or equal to 0 ****`

- user entering eof (interactive mode only):
  interactive mode query,
  input file name

  display
  `**** EOF entered - program terminating ****`

  output file name

  No error message, ignored and user forced to enter valid name

  invalid menu option

  Specified in first option

  anywhere else

  `**** EOF ignored - to abort the program type Ctrl-c ****`
  Forces re-entry of valid data

Invalid list position error, i.e. the specified list position is > 0 but also > the maximum number of items on the list.

display
`**** Player number must be between 1 and max ****`
The command handler should force the user to enter a valid player number.

It is not possible to:

In each case, the corresponding command handler should display the following message, where *name* is the actual player name and exit without further action.

- move a player currently on the Injury Reserve list to the Injury Reserve list,

  display
  `**** *name* is already on the injury list ****`

- activate a player currently on a team other than the Injury Reserve list

  display
  `**** *name* is already on an active roster ****`

- update a player's points if they are on the Injury Reserve list.

  display
  `**** *name* is on injury reserve so can't be updated ****`

Design and Implementation

The Linked List

The program must use **one** dynamically allocated singly linked list to store the "Olympic Teams" list. The data part of each node must be a C++ struct named PlayerNode containing

- the player name
- the player team (`'C' = Canada, 'F' = Finland, 'U' = USA, 'I' = Injury Reserve`)
- the player position (`'f' = forward, 'd' = defenceman, 'g' = goaltender`)
- the player goals
- the player assists
- the player goals against

As described above, the list must remain **ordered by team** at all times. The "activate" and "move" commands insert nodes at the end of the given team sub-list. Because of the available list operations, described below, the update command will also result in a player changing positions on the team list. Review the output from the activate and update commands, above. Note that the player's list position is NOT stored in the player's node, it is determine in the processing of each command.

The list is composed of nodes that have the following typedef and struct definitions. A `teamList.h` header file is provided that contains these items and **the given items may not be changed in any way!**

```
typedef struct PlayerNode *PlayerPtr;

struct PlayerNode
{
   char name[MAXNAMELEN+1];
   char team;
   char position;
   int goals;
   int assists;
   int goalsAgainst;
   PlayerPtr next;
};
```

Your program **must** be decomposed into at least two modules:

1) a main module, called `teams.cpp`, which will contain the main function, a ***command handler function*** for each menu operation and any additional helper functions used to logically decompose the problem.
2) a linked list module, consisting of `teamList.cpp` and `teamList.h`, containing all of the functions that operate on the team list, as well as the above items.

The Main Module (`teams.cpp`)

The main program needs to perform the following tasks:

- create a head pointer to the team list and initialize it to NULL
- read the filename of the team roster file from the user
- process the team file, which consists of:
  - seeing if the filename is not empty
  - if the team file can be successfully opened then it is to be read and a team list created; otherwise the program should gracefully terminate
  - if the filename the user enters is empty the program should continue with an initially empty list. ~~The user should be informed of this situation.~~
- process user requests
  - print the menu
  - read the user's request
  - call the corresponding **command handler function** to process the user's request. The command handler function will read any additional info required to successfully process the request, i.e. when moving a player to the Injury Reserve the corresponding command handler function will read the list number from the user. Any and all output will be generate by the command handler function.
  - this will continue until the user requests to quit the program.
- write the linked list to a team file
  - read the filename from the user
  - attempt to open the file
    (Remember that failure to open an output file will be due to either: lack of permission to the directory, the hard drive being full or hardware failure. Since we are currently unable to determine which of these caused the failure, the program should simply inform the user of this failure and omit the next step.)
  - write the contents of the linked list to the file
- clean up the linked list
  - While the dynamically allocated nodes that compose the linked list would be reclaimed by the system upon program termination, it is always good practice to explicitly return the nodes to the heap! This will be done by a call to an appropriate function from the linked list module.

While these actions have been specified to be done in the main module, this does not mean they all should be done in one function! As always, tasks should be broken up into functions to reduce duplication and to make your code as expressive as possible. As previously stated, the code to process each menu command should be placed in a separate function, called a command handler. In addition, the following two helper functions **must** be used:

- `int load (istream& infile, PlayerPtr& head);`

  This function reads the contents of an open input file stream (infile) and creates a linked list in memory. The input stream must be successfully opened by the caller. Because the function is loading the list it is assumed that no prior list exists and that the head variable is passed in

pointing to NULL. When the load function completes head should either point to NULL or to the first node of the list that has been created. The information for each player is read from the file and stored in a player node which is added to the end of the existing linked list. The function returns the number of items added to the list.

- `void write (ostream& outfile, PlayerPtr head);`

  This function writes the team list to the given file in the same format as the input file. The output stream must be successfully opened by the caller.

Note: the istream and ostream types in the two above headers are NOT typos. These types are generalizations of both iostreams and fstreams – this means that variables of either types, i.e. cin/cout or files will be accepted by these variables. Thus, when constructing these function you can use cin and cout for testing purposes. When you change to using files you will declare variables of either ifstream or ofstream.

The operations required in the `team.cpp` module **must not** operate directly on the linked list, i.e. removing a node from the list. Thus, these operations **can only** work on the teams list using the list operations, which are described in the next section.
The **only exception to this restriction** is that the client program can **output** needed player fields.
The operations described in this section must be created and used appropriately by the `teams.cpp` module.
Failure to follow these two requirements will result in a significant grade loss.

The List Module (`teamList.h & teamList.cpp`)

As previously stated a teamList.h header file is provided that contains the specified node definition and node pointer typedef . It also contains the prototypes specified below and some additional constants and typedefs that you might find useful in implementing operations in both the `teams.cpp` and `teamList.cpp` modules. This header file will be included in the implementation file (`teamList.cpp`) and client program (`teams.cpp`).

The implementation file (`teamList.cpp`) will include function definitions for the visible functions specified in the header file, but will also contain function definitions for any helper functions that are useful for implementing the visible functions.

The following visible function definitions must be created:

- ```
  PlayerPtr createPlayer (const char name[], char team, char position,
                          int goals, int assists, int goalsAgainst);
  ```

  This function dynamically creates a new `PlayerNode`, initialize the six fields with the given parameter values, and returns a pointer to the new loaded node.
  ```

- `void search(PlayerPtr head, int itemNumber,`
  `PlayerPtr &player, bool &found);`

  This function searches the team list by position, which is an integer that must be > 0. If this position exists in the team list, search will return a pointer to the node in that position and the Boolean found will be set to true; otherwise, the pointer will be set to NULL and found will be set to false.

- `void insertPlayer(PlayerPtr &head, PlayerPtr player);`

  This function adds a player to a sorted teams list, such that the list remains sorted by team with this new player added to the end of team they are on (a field in the PlayerNode structure). It will be assumed that the user will NOT attempt to put a duplicate item in the list. Hint: check out the TEAMCHARS array in the teamlist.h file - the teams are almost stored alphabetically in the list, except for the Injury Reserve list. Therefore, it will probably be easier to covert the team letter to an integer and this can be done using this array.

- `void removePlayer(PlayerPtr &head, PlayerPtr player);`

  This function will **remove the specified player** from the team list the and will leave the list sorted. Upon completion the PlayerPtr will still point to the node that has been removed from the list. If the node is to be deleted, i.e. returned to the heap, it is the responsibility of the client program. It is assumed that the function will be called with PlayerPtr pointing to a player that is currently on the list.

- `void displayTeams(PlayerPtr head, ostream &out);`

  This function writes a teams list to the specified output stream with a header for each team and with the counter value for each node, starting at 1, i.e. as shown in the above examples. The team header should only be output if there are players from this team to be printed, for example if a teams list only contained players from Canada and the USA then the "Team Finland:" header would not be printed.

- `bool containsActivePlayers (PlayerPtr head);`

  This function checks whether there is at least one active player on the list. If there is at least one active player it returns true; otherwise it returns false.

- `void destroyList (PlayerPtr &head);`

  This function deallocates all the nodes on the list, and sets head to NULL.

In the discussion of Linked Lists a catalogue of operations was discussed. A number of these should probably be included in the library, but should not be visible to the client since they should only be used by the visible functions. As you are designing the visible functions, if it does not become

apparent why these helper functions are useful, you should talk with either your professor or an Instructional Assistant.

CRITCAL IMPLEMENTATION DETAILS

The main program is menu-driven; however, testing a menu-driven program can be ***extremely*** annoying, as you need to manually type each command and all of the corresponding data each time you run the program after making any changes to the code. In order to avoid this, one technique is to build the program so that it works in either **interactive mode**, i.e. the user enters input, or **batch mode**, i.e. the input can be read from a file. To do this, the first thing the program does is prompt the user whether the program is to be run interactively or not. If the user responds with a 'y' then the program will generate prompts and read user input. If the response is an 'n' then no prompts will be generated and the data can be read from a file using command line redirection. The input file will have each input item in a file on a separate line.

The user's response to how to run the program, interactively or not, should result in a Boolean value - check out the `ioutil` routines. Then, each read that uses an `ioutil` routine should be enclosed in an if-then-else, with one option being interactive with a prompt and the other option being reading from the file with no prompt. A possible example, where the user's answer to the y/n question has set the Boolean variable `interactive` accordingly, would be:

```
if (interactive)
{
   readInt("Enter a player position: ", place, eof);
}
else
{
   readInt("", place, eof);
}
```

When you get this assignment, your knowledge of linked lists will be insufficient to complete the entire assignment. Therefore, you should create the main program, i.e. the main function, the command handler functions and any other helper functions you determine are required. You can either create the linked list module with stub functions for the required operations or simply omit calling them in the command handler functions. Initially write the program to be interactive. Then once you know the order of inputs, you can create an input file to do a specific command(s) and then alter each data read as shown above.

Testing

All functions should be thoroughly tested before they are used. In order to get you to follow this practice you are required to use `GoogleTest` to create a test module that completely tests the functionality of `insertPlayer()`. Your ASSERTs/EXPECTs will use the `search()` function.

Submission & Grading

Detailed submission instructions will be given a day or two before the due date.