NFL Standings

Programmer Manual

By: Jeffrey Marron for B. Streller’s Fall 2015 CS132 class

Contents:

1. cs132\_P2\_driver.cpp

1. Print Scores

2. Add a Game

3. Add a Team

4. Delete a Team

5. Display Menu

6. Exit

1. standings.h
2. STL list

I – cs132\_P2\_driver.cpp

First, a standings object is created and the title screen is displayed. Then the getFileName function gets the inFile name from the user. The getInfo function then runs, retrieves the data from the inFile, and places the data into list nodes inside the teamlist. The main menu is then displayed and the program enters a do-while loop where the user can enter further input through a switch-case operated command interface.

1 – Print Scores

This option runs the print function to display the contents of the list to the console. Cclear is then called to clear the input buffer.

2 – Add a Game

This option runs the AddAGame function to update the list of teams with appropriate info from a game. It then calls the sortTeam function to update the order of the team standings after the changes made from the AddAGame function. The writeFile function is then called to update the changes to the outFile. Cclear is then called to clear the input buffer.

3 – Add a Team

This option calls the addTeam function to create a new teamlist list node of team data which is input from the user. It then calls the sortTeam function to update the order of the team standings after the changes made from the addTeam function. The writeFile function is then called to update the changes to the outFile. Cclear is then called to clear the input buffer.

4 – Delete a Team

This option calls the deleteTeam function to delete a list node of team data. The writeFile function is then called to update the changes to the outFile. Cclear is then called to clear the input buffer.

5 – Display Main Menu

This option runs the displayMenu function to display the main menu options on the console. Cclear is then called to clear the input buffer.

6 – Exit

This option exits the program.

---------------------------------------------------------------------------------------------------------------------

II – standings.h

The standings class inherits the team struct. It defines the linked list of team objects using the stl list. Standings.h also includes various functions used to get and manipulate the linked list data.

-getInfo

This function reads data from the inFile using the getline function and places the data into string str. A subroutine runs which evaluates str for white spaces. If str contains good data then it is placed into the string stream iss and then parsed into appropriate data containers of the team struct. After an entire line is parsed, the list push back function is called into a new list node. This sequence loops until all of the data from the inFile is retrieved and parsed.

-print

This function prints the date and then iterates through the teamlist and prints out the data onto the console output. It places the appropriate division headings by examining the division number of each list node and int check. The division number starts out as 1 and check as 2. As soon as the division number reaches 2 & is equal to check, the next division header is printed and check iterates. This process continues until all of the divisions are printed out.

-sort

This function sorts the list by ascending division, then sorts each division by descending percentage, and then sorts the team names by ascending alphabetical order. This function uses the STL sort algorithm through separate overloaded function calls for compareDiv, comparePCT, and compareName.

-getFileName

This function takes user input from the getline and places it into the inFilename. It evaluates for valid input and will return the inFilename to the program for use in other functions if a valid file name is entered.

-writeFile

This function iterates through the linked list and writes the node data to an outFile using the inFilename as a parameter. It also calculates the percentages and updates it before writing to file.

-addTeam

This function asks the user for several pieces of input data. The other data members are then appropriately calculated. All the data members are then placed into a list node using the push back function.

-AddAGame

This function asks the user to enter two team names. If the names exist in the list then the division values are placed into the CurrentHomeTeam and CurrentRoadTeam respectively. It then asks the user if the home team won, lost, or tied. The user input is passed into int choice and used to operate a switch case which compares both teams divisions and then changes their wins, losses, road wins, road losses, division wins, and division losses.

-deleteTeam

This function asks the user to input a team name. It then evaluates the teamName from the list of nodes and deletes the appropriate list node with the list erase function.

---------------------------------------------------------------------------------------------------------------------

III – STL list

The main difference between singly and doubly linked lists is the use of a link to a previous node through a previous node pointer. The STL list utilizes doubly linked lists. Its data node structure shows how this is implemented in both the generalized base template definition (lines 435-445 Microsoft Visual Studio 12.0\VC\include\list) and the specialized template definition (447-457) with the use of \_Voidptr \_Prev and \_Nodeptr \_Prev respectively.

By having both next and previous pointers, a doubly linked list can be traversed both forwards and backwards. That being said, the functions in this program which require iteration through the list are all implemented to start at the beginning and traverse forwards (AddAGame, writeFile, print, and deleteTeam) and gain no benefit from being doubly linked. Furthermore, functions which add data to the team and don’t require iteration (getInfo, addTeam) utilize the push back function which also does not benefit from being doubly linked. Sorting the team list is done through the STL list::sort function (1706-1751) which utilizes splice & mergesort but again the team list does not require nor does it gain any benefit from being doubly linked while using this sorting algorithm. This program does however benefit from being doubly linked in its deleteTeam function which uses the list::erase function. In a singly linked list, deleting a node requires a temporary pointer where the temp gets the current, the previous link gets the current link, the current gets the current link, and then the temp is deleted. With a doubly linked list the previous link just needs to get the next node (in the STL list::erase this can be seen in lines 1467-1470).