Doubly Linked Circular List
Lab 6

Prepared by Matt Bennett
for ELEC 3150
Professor Carpenter - Fall 2016

I - Introduction

The goal of this lab was to create a doubly linked circular list from a provided text file. In addition to requiring a basic understanding of linked lists, this lab required knowledge of C/C++ structures, functions, deletion, and pointers.

II - Procedure

The basic procedure for creating and maintaining a circular linked list was created using a head pointer, a tail pointer, and a series of structures containing previous entry and next entry pointers. At the start of the program, entries are read into the program from a text file with each unique name populated by the entry function.

The entry function generates pointers based on the current status of the linked list. In cases where there are not yet any structures in the linked list, located at the head pointer, a new structure is instantiated. Both the head and tail pointers are updated with the address of the newly created structure and the structure's previous and next pointers are set to link to itself. When structures are already stored at the header pointer, the list is iterated over using the structures next pointers. The structure's name is compared to the current name during each iteration so that duplicately named structures are not created and each structure is updated to reflect the addition of the newly input information for that particular name. If no matching entry is found before reaching the tail, the tail is saved as a temporary entry and a new entry is created. In order to place the new entry at the end of the preexisting list, the tail is updated to point to the new entry. The new entries previous pointer is then set to the saved temporary entry and the next pointer is set to the head pointer. Finally, the head entry is updated so that the previous pointer points to the newly created tail entry.

To remove a particular entry from the linked list, the previous and next entries must be updated to 'close the gap' caused by removing the entry. Additionally, in the case of removing a head or tail entry, the head or tail pointer respectively, must be updated to prevent losing track of the beginning and ending entry in the list. Once the entry to be deleted is located, it is saved to temporary pointer. The temporary pointer is then compared to the head and tail pointers, updating the head to be the next pointer or updating the tail to be the previous pointer when necessary. The entry before the deleted item then has its next pointer set to the entry after the deleted item and the entry after the deleted item has its previous pointer set to the entry before the deleted entry. This process essentially removes the item to be deleted and 'closes the gap' of previous and next pointers created. Once the entry is fully separated from the linked list, the entry and the temporary pointer are deleted.

III - Results

```
bennetts4@turing:~/Lab06$ ./Lab06
Name: Kurt Warner Number of wins: 1 Years: 2000
Name: Trent Dilfer Number of wins: 1 Years: 2001
Name: Tom Brady Number of wins: 4 Years: 2002 2004 2005 2015
Name: Brad Johnson Number of wins: 1 Years: 2003
Name: Ben Roethlisberger Number of wins: 2 Years: 2006 2009
Name: Peyton Manning Number of wins: 1 Years: 2007
Name: Eli Manning Number of wins: 2 Years: 2008 2012
Name: Drew Brees Number of wins: 1 Years: 2010
Name: Aaron Rodgers Number of wins: 1 Years: 2011
Name: Joe Flacco Number of wins: 1 Years: 2013
Name: Russell Wilson Number of wins: 1 Years: 2014
Select an option:
 1: search list
 2: remove entry
 3: print list
 4: exit
```

Figure 1: Initial population of linked list and user option menu.

```
Entry to remove [first last]: Kurt Warner
Select an option:
 1: search list
 2: remove entry
 3: print list
 4: exit
3
Name: Trent Dilfer Number of wins: 1 Years: 2001
Name: Tom Brady Number of wins: 4 Years: 2002 2004 2005 2015
Name: Brad Johnson Number of wins: 1 Years: 2003
Name: Ben Roethlisberger Number of wins: 2 Years: 2006 2009
Name: Peyton Manning Number of wins: 1 Years: 2007
Name: Eli Manning Number of wins: 2 Years: 2008 2012
Name: Drew Brees Number of wins: 1 Years: 2010
Name: Aaron Rodgers Number of wins: 1 Years: 2011
Name: Joe Flacco Number of wins: 1 Years: 2013
Name: Russell Wilson Number of wins: 1 Years: 2014
```

Figure 2: Removal of head list entry and the resulting list.

```
Entry to remove [first last]: Russell Wilson
Select an option:
 1: search list
 2: remove entry
 3: print list
 4: exit
3
Name: Trent Dilfer Number of wins: 1 Years: 2001
Name: Tom Brady Number of wins: 4 Years: 2002 2004 2005 2015
Name: Brad Johnson Number of wins: 1 Years: 2003
Name: Ben Roethlisberger Number of wins: 2 Years: 2006 2009
Name: Peyton Manning Number of wins: 1 Years: 2007
Name: Eli Manning Number of wins: 2 Years: 2008 2012
Name: Drew Brees Number of wins: 1 Years: 2010
Name: Aaron Rodgers Number of wins: 1 Years: 2011
Name: Joe Flacco Number of wins: 1 Years: 2013
```

Figure 3: Removal of tail list entry and the resulting list.

```
Select an option:
 1: search list
 2: remove entry
 3: print list
 4: exit
1
Select an option.
 1: name
 2: year
 3: minimum number of wins
 4: exit menu
3
Enter search term: 2
Name: Tom Brady Number of wins: 4 Years: 2002 2004 2005 2015
Name: Ben Roethlisberger Number of wins: 2 Years: 2006 2009
Name: Eli Manning Number of wins: 2 Years: 2008 2012
```

Figure 4: Search by number of wins.

```
Select an option:
 1: search list
 2: remove entry
 3: print list
 4: exit
4
bennetts4@turing:~/Lab06$
```

Figure 5: Delete entire list and exit program.

IV - Analysis

The linked list functioned as expected with list population tested as shown in Figure 1, head and tail entry removal tested as shown in Figures 2 and 3, searching tested as shown in Figure 4, and program exit and list deletion tested as shown in Figure 5. Although the program and circular doubly linked list functioned as expected and undoubtedly has benefits in many scenarios, the circular nature of the data structure in this particular case seems to be unnecessarily cumbersome as none of these circular functionalities were implemented.

V - Conclusions

The algorithm functions as expected for the populating, removing, and searching entries in the linked list. Removal of entries properly updates the list to maintain the linked list connectivity created by the various pointers in the data structure.