EECS 268: Spring 2009
Laboratory 4: Another Linked List ADT

Due: 11:59:59 pm the day before your lab meets the week of March 9

Lecture Topics: Doubly-Linked Lists ADT and list operations
Lab Topics: C++ Pair Class

1. Introduction

Due to the initial success of your movie inventory system, CubeCompressor, a fledgling online movie rental company, has asked you to implement several new features for their much-hyped online debut. First, CubeCompressor wants your system to include the first and last names of all actors and actresses in each movie. These names will be stored either in a predefined order or sorted by the actor's last name and first name. Second, CubeCompressor wants to be able to very quickly print the actors' names associated with a given movie, either in the stored order or in reverse order. Third, CubeCompressor wants your system to be able to print the name of every movie that a given actor has performed in.

This lab will build upon and extend the movie database system you created in lab 3. In addition to processing several new commands from the input file, you will also be implementing a new doubly-linked list in your existing Movie class. (Note: You *must* turn in a working lab 3 for grading before starting on this lab. Completing lab 4 cannot be used in lieu of completing and submitting lab 3!)

The overall purpose of this lab is to understand and be able to implement the doubly-linked list abstract data type (ADT), and to demonstrate top-down design with stepwise refinement in developing software systems and the use of good programming style to improve clarity and readability of programs.

2. Getting Started

Most of the changes to your existing movie database system will be in the Movie class. In order to quickly print the actors' names in forward and reverse order, you will be implementing a doubly-linked list and maintaining a pointer to both the head and the tail of the list. Following are the details of your changes to the Movie class:

2.1. Movie

This class will define an ADT containing the pointers and structures necessary to hold a doubly-linked list of actors' names. This will include both a Node structure defining the doubly-linked list node and a pointer to the head and tail of the list. To simplify storing and comparing the actors' names, you will be taking advantage of the C++ pair class within your linked list. Therefore, the node structure of the doubly-linked list will be storing objects of type `pair<string, string>` as the data item of each list node.

Note: Once again, rather than repeating `pair<string, string>` throughout your ADT, you will need to create a new data type named NodeItemType using typedef. All references to the data type of your node item should then use NodeItemType.

The Movie class will be expected to include the following functionality. It will be up to you to define the function prototypes and to determine the best way to implement this functionality.

input and output – The overloaded input operator should now be responsible for reading in the list of actors' names and inserting them correctly into the doubly-linked list. Inserting the actors' names will be done using one of the following insert functions. The overloaded output operator should now be responsible for printing out the list of actors' names in both forward and reverse order.

insert first – This function should insert a new actor's name at the head of the doubly-linked list.

insert last – This function should insert a new actor's name at the tail of the doubly-linked list.

insert sorted – This function should traverse the doubly-linked list, inserting a new actor's name into its correct sorted position (assuming that the linked list is sorted by last name and then first name).

remove – This function should remove the specified actor's name from the doubly-linked list. If the actor's name does not exist in the linked list, a MovieException should be thrown.

contains – This function should search the doubly-linked list for the specified actor's name, returning true if it is found and false if it is not found.

sort – This function should sort the actors in the doubly-linked list by last name and first name. The following source code has been provided as a framework for this sort function. However, it will be up to you to complete the bubble sort implementation by providing a swapNext function that swaps the specified node in the doubly-linked list with its following node (updating all pointers, including head and tail, as necessary).

```
void Movie::bubbleSort()
{
  // continue sorting while items in the list are being moved
  while (bubbleSortR(head));
}

bool Movie::bubbleSortR(ListNode* &node)
{
  // base case: list is empty or we've reached the last node in the linked list
  if (node == NULL || node->next == NULL)
  {
    return false;
  }
  // recursive case: sort in ascending order
  else if (node->item > node->next->item)
  {
    // Swap current node with next node
    swapNext(node);

    // Continue sorting, but always return true because a swap has occurred
    bubbleSortR(node->next);

    return true;
  }
  // recursive case: no swap is necessary, but return true if swap occurs later in the list
  else
  {
    return bubbleSortR(node->next);
  }
}
```

2.2. MovieDatabase

Your changes to the MovieDatabase class will consist mostly of thin wrapper functions around the new Movie functionality, such as **sort** and **remove actor**. However, the MovieDatabase class will also be expected to include the following functionality. It will be up to you to define the function prototypes and to determine the best way to implement this functionality.

print actor – This function should search each movie in the database for the specified actor. The function should print a numbered list of the movie names that the actor has appeared in. This list should number from 1 to *x*, where *x* is the number of movies the specified actor is in. (Therefore, these numbers are not related to index of each movie in the linked list.) If the actor was not found in any movies, this function should print "None."

3. Implementation Details

As in lab 3, exceptions are to be used for all error handling procedures. The following new situations are considered to be errors in this lab.

- Attempting to sort the actors' names in a nonexistent movie.
- Attempting to remove an actor from a nonexistent movie.
- Attempting to remove an nonexistent actor from a movie.

If an exception is thrown, an error message should be printed out indicating the cause of the error, and the program should continue processing input.

4. Input & Output

Your program will be expected to read input from the first file specified on the command line and to write output to the second file specified on the command line. Your program should display an error message and exit immediately if the expected command-line parameters are not provided. Following is an example of how your program will be run from the command line:

```
$ ./main input.txt output.txt
```

Input will be a series of commands, some of which are followed by data to be used with that command. In addition to the commands included in lab 3, here are the new or modified commands your program must handle:

insert – This command will be followed by an integer representing the index within the list at which the new movie will be inserted (remember, the list will use 1-based indexing). The second line will contain information about the new movie. The values on this line will be delimited by the '|' character, and will be in the following order:

```
<name>|<year>|<rating>|<description>|
```

The insert command will now be followed by two additional lines. The third line will include include the number of actors and the method that these actors will be inserted into the linked list. The fourth line will include the names of the actors (delimited by the '|' character). The insertion method will be either <u>first</u> (each name is inserted at the front of the list), <u>last</u> (each name is inserted at the back of the list), or <u>sorted</u> (each name is inserted into its sorted position in the list).

```
<insertion method> <number of actors>
<first name> <last name>|<first name> <last name>|...|<first name> <last name>|
```

Processing this command should involve creating a new Movie instance containing the appropriate information and attempting to add it to the linked list at the denoted position. Here is a full example of the insert command:

```
insert 1
Gamera|1965|G|A military plane crashes in the Arctic, awakening giant turtle-monster Gamera.|
sorted 5
Eiji Funakoshi|Harumi Kiritachi|Michiko Sugata|Yoshio Yoshida|Daihachi Kita|
```

sort – This command will be followed by an integer representing the index of the movie whose actors' names will be sorted. Processing this command should sort the doubly-linked list of actor names in the specified movie by last name and first name. Example:

```
sort 2
```

remove-actor – This command will be followed by an integer representing the index of the movie in which to delete the specified actor. The command will follow this format:

    remove-actor <index> <first name> <last name>

Processing this command should attempt to delete the actor within the specified movie. Example:

```
remove-actor 2 Virginia Aldridge
```

print-actor – This command will be followed by the name of the actor to print. The command will follow this format:

    print-actor <first name> <last name>

Processing this command will search each movie in the database for the specified actor, print either the name of each movie the actor was in, or "None" if no movies were found. Example:

```
print-actor Virginia Aldridge
```

<u>4.1. Sample Input File:</u>

```
insert 1
Gamera|1965|G|A military plane crashes in the Arctic, awakening giant turtle-monster Gamera.|
sorted 5
Eiji Funakoshi|Harumi Kiritachi|Michiko Sugata|Yoshio Yoshida|Daihachi Kita|
insert 2
Santa Claus Conquers the Martians|1964|G|In this children's movie, Martians abduct Santa...|
first 2
John Call|Leonard Hicks|
insert 1
High School Big Shot|1953|PG|A high school dweeb plans a heist to win the favor of a pretty classmate.|
last 4
Tom Pittman|Virginia Aldridge|Howard Veit|Malcolm Atterbury|
print-all
insert 7
The Beast of Yucca Flats|1961|R|A defecting Russian scientist turns into a monster.|
last 5
Douglas Mellor|Barbara Francis|Bing Stafford|Larry Aten|Linda Bielema|
sort 2
print 2
remove 2
print-all
insert 3
Parts: The Clonus Horror|1979|PG-13|Clones are being bred to serve as a source of replacement organs for the
wealthy.|
sorted 6
```

```
Tim Donnelly|Paulette Breen|Dick Sargent|Keenan Wynn|Zale Kessler|Virginia Aldridge|
swap 1 3
print-all
print-actor Virginia Aldridge
remove-actor 1 Virginia Aldridge
remove-actor 2 Virginia Aldridge
remove-actor 3 Virginia Aldridge
print-actor Virginia Aldridge
exit
```

4.2. Sample Output File:

```
Current Movies in the Database:

Position: 1
------------
High School Big Shot
	year: 1953
	rating: PG
	desc: A high school dweeb plans a heist to win the favor of a pretty classmate.
	actors (A-Z): Tom Pittman, Virginia Aldridge, Howard Veit, Malcolm Atterbury
	actors (Z-A): Malcolm Atterbury, Howard Veit, Virginia Aldridge, Tom Pittman

Position: 2
------------
Gamera
	year: 1965
	rating: G
	desc: A military plane crashes in the Arctic, awakening giant turtle-monster Gamera.
	actors (A-Z): Eiji Funakoshi, Harumi Kiritachi, Daihachi Kita, Michiko Sugata, Yoshio Yoshida
	actors (Z-A): Yoshio Yoshida, Michiko Sugata, Daihachi Kita, Harumi Kiritachi, Eiji Funakoshi

Position: 3
------------
Santa Claus Conquers the Martians
	year: 1964
	rating: G
	desc: In this children's movie, Martians abduct Santa...
	actors (A-Z): Leonard Hicks, John Call
	actors (Z-A): John Call, Leonard Hicks

ERROR: Inserting at position out of range

Movie at position: 2:
Gamera
	year: 1965
	rating: G
	desc: A military plane crashes in the Arctic, awakening giant turtle-monster Gamera.
	actors (A-Z): Eiji Funakoshi, Harumi Kiritachi, Daihachi Kita, Michiko Sugata, Yoshio Yoshida
```

```
        actors (Z-A): Yoshio Yoshida, Michiko Sugata, Daihachi Kita, Harumi Kiritachi, Eiji Funakoshi

Current Movies in the Database:

Position: 1
------------
High School Big Shot
        year: 1953
        rating: PG
        desc: A high school dweeb plans a heist to win the favor of a pretty classmate.
        actors (A-Z): Tom Pittman, Virginia Aldridge, Howard Veit, Malcolm Atterbury
        actors (Z-A): Malcolm Atterbury, Howard Veit, Virginia Aldridge, Tom Pittman

Position: 2
------------
Santa Claus Conquers the Martians
        year: 1964
        rating: G
        desc: In this children's movie, Martians abduct Santa...
        actors (A-Z): Leonard Hicks, John Call
        actors (Z-A): John Call, Leonard Hicks

Current Movies in the Database:

Position: 1
------------
Parts: The Clonus Horror
        year: 1979
        rating: PG-13
        desc: Clones are being bred to serve as a source of replacement organs for the wealthy.
        actors (A-Z): Virginia Aldridge, Paulette Breen, Tim Donnelly, Zale Kessler, Dick Sargent, Keenan Wynn
        actors (Z-A): Keenan Wynn, Dick Sargent, Zale Kessler, Tim Donnelly, Paulette Breen, Virginia Aldridge

Position: 2
------------
Santa Claus Conquers the Martians
        year: 1964
        rating: G
        desc: In this children's movie, Martians abduct Santa...
        actors (A-Z): Leonard Hicks, John Call
        actors (Z-A): John Call, Leonard Hicks

Position: 3
------------
High School Big Shot
        year: 1953
        rating: PG
        desc: A high school dweeb plans a heist to win the favor of a pretty classmate.
```

```
          actors (A-Z): Tom Pittman, Virginia Aldridge, Howard Veit, Malcolm Atterbury
          actors (Z-A): Malcolm Atterbury, Howard Veit, Virginia Aldridge, Tom Pittman

Found actor Virginia Aldridge in:
          1. Parts: The Clonus Horror
          2. High School Big Shot

ERROR: Actor not found.

Found actor Virginia Aldridge in:
          None.
```

5. Grading Criteria

Grades will be assigned according to the following criteria:

20 **Main Function & MovieDatabase Class**
- ⇒ Process new (and modified) commands in Main.
- ⇒ Implement wrapper functions and new functionality in MovieDatabase.

40 **Movie Class & Doubly-Linked List**
- ⇒ Correct definition of doubly-linked list.
- ⇒ Correct implementation for inserting, searching, and deleting in a doubly-linked list.
- ⇒ Completed implementation of bubble sort function.
- ⇒ Correct input and output processing for new Movie class data.

10 **Exception Handling**
- ⇒ Exceptions are thrown for each case described above.
- ⇒ All exceptions are caught and recovered from in the main function.

20 **Documentation**
- ⇒ Javadoc comments for all files, classes, and functions.
- ⇒ Internal documentation (especially recursive functions).

10 **Programming Style & Output Formatting**
- ⇒ Use standard paragraphing conventions and object-oriented design practices.

100 **Total**