Department of Electrical and Computer Engineering
University of Puerto Rico
Mayagüez Campus

ICOM 4035 – Data Structures
Fall 2001

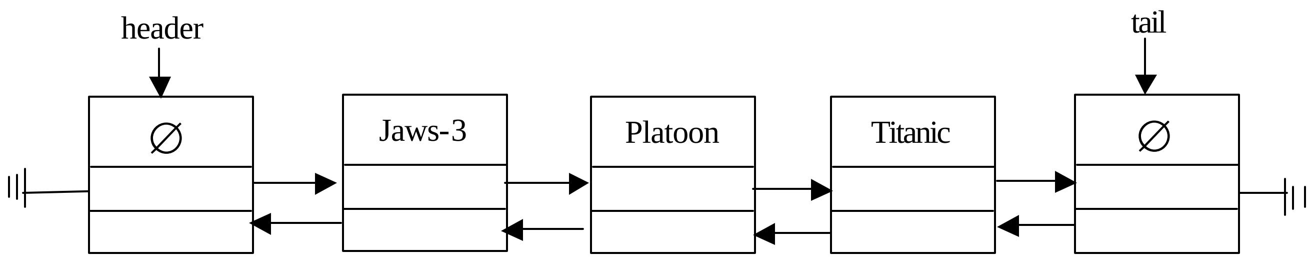
Project #2: Movie Database Application using Linked Lists
Due Date: 11:59 PM-November 20

Objectives

1. Understand the design, implementation and use of a linked lists class container.
2. Gain experience implementing applications using layers of increasing complexity and fairly complex data structures.
3. Gain further experience with object-oriented programming concepts, specially templates and operator overloading

Overview

You will implement and test an application that acts as main-memory database containing information about movies. All the records with movie information will be stored in a **sorted** doubly linked class container. The sorting for the movie record will be based on the movie title. For simplicity we will assume movie titles consisting on either a one word, or multiple words separated by a dash - . For example: Platoon, Titanic, Jaws-3. The following diagram illustrates a general level organization of a movies database for these three movies.



In reality, each node will not have simply the movie title, but a record that has the following information:

1. Movie Title
2. Movie year of release
3. Duration of the movie (minutes)
4. Rating of the movie (must be one of NR, G, PG, PG-12, R, NC-17)
5. Linked List with the name of the actors

You will implement the following operations for the movie database application, using the a class called the DataManager:

1. Add movie – add a new movie record to the movie database.
2. Delete movie – deletes a movie record from the movie database.
3. Add cast – adds a new cast member name to a movie
4. Delete cast – deletes a cast member name (if present) from a movie
5. Find movie – finds the information record for a movie
6. Find movies by actor name – finds all movies by a given actor. Returns a new sorted doubly linked list with them.
7. Find movies by rating – finds all movies with a given rating. Returns a new sorted doubly linked list with them.
8. Find movies by year – find all movies made in a given year. Returns a new sorted doubly linked list with them.
9. Find movies by time period – finds all movies in a time period (e.g. 1980-1985). Returns a new sorted doubly linked list with them.
10. Print movies in order – prints all the movies in the sorted order.
11. Print movies in reverse order – prints all the movies in the reverse sorted order

NOTE: You must keep the movies in sorted order at all times.

<u>Sorted Doubly Linked List Class Container</u>

You **must** implement a sorted doubly linked-list class container to keep the information about the movie records, and also to keep the name of the actors within a movie. The sorted doubly linked list is a template that receives two parameter types:

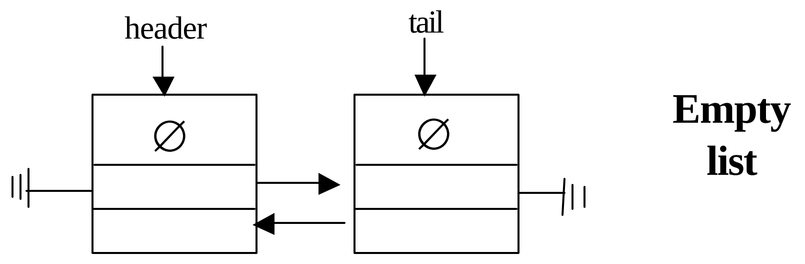
1. ListData – the object to be stored in the list. It is assumed by the templates that a ListData type will implement a function called **get_key()** that returns an object of type DataKey (explained below). In addition, the ListData data-type must overload the relational operators = =, !=, <, <=, >, >=. In this project you will use classes Name and Movie in this role of ListData. These classes are explained below.
2. Data Key – represents an identifier for the object stored in the list. It should be returned from the a call to method **get_key()**.

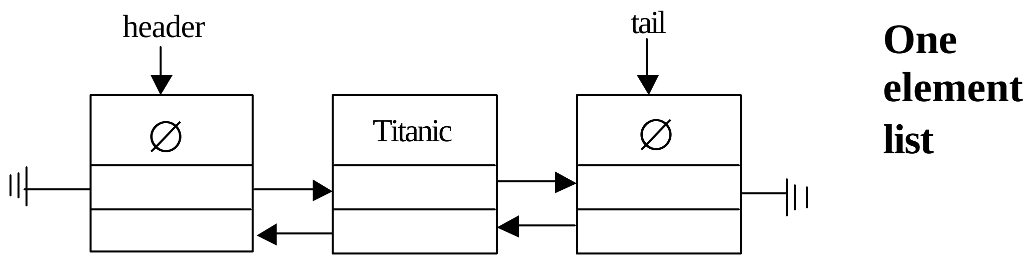
The sorted doubly linked list has to special nodes delimiting the node with the list data.

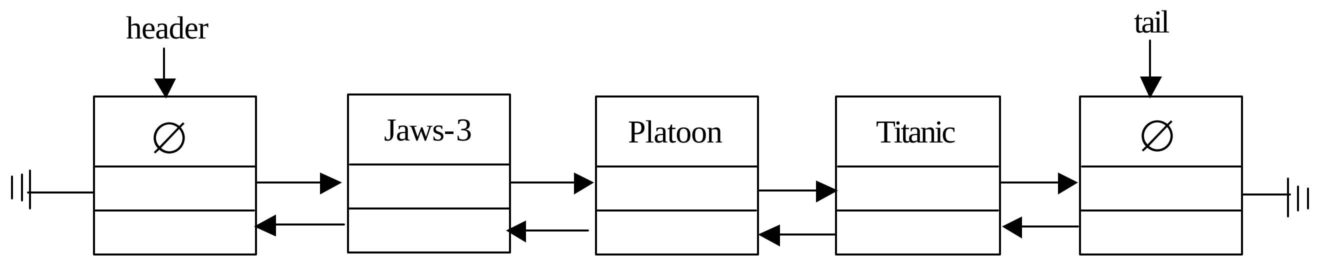
1. header node– has no data and points to the first element in the list. If the list is empty, it points to the tail node.
2. tail node – has no data and points to the last element in the list. If the list is empty, it points to the header node.

Each node in the list has a **next** field that points to the next node in the list. Similarly, each node has a **prev** field which points to the previous node in the list. Finally, the data in each node is stored in a field called **data**.

The following diagram shows several cases of sorted doubly linked lists:







Remember that you will use the sorted doubly linked list for two purposes:

1. Implement the list of movie records in the database.
2. Implement the list of actor names in a movie record.

NOTE: For the sorted linked list class container, you MUST follow these guidelines.

1. The erase operator must do an in-place removal of the target node. It cannot copy nodes to a new list, skipping the node that you want to delete. Programs that deviate from this direction will not be considered a running program.
2. The destructor cannot create memory leaks.
3. The copy constructor must create a deep copy of the lists.
4. The insert operation must put a new node in the appropriate sorted order. If your lists are unordered, your program will not be considered a running program.

<u>Name Class</u>
The name class will represent the name of an actor in a movie. It will have two private fields, the actor's last name and the actor's first name. **You will be given the C++ for this class**.

Name List Class

The name list class represents a sorted list of names. This classes uses the sorted doubly linked list to maintain the list of names. **You will be given the C++ for this class.** (Except for the linked list code).

Data Manager

You must implement a Data Manager class that will maintain the list of movies. The Data Manager has the list of movies a private member. The Data Manager provides the interface to perform all the maintenance operations on the movie database. These operations were mentioned at the beginning of this document.

Distribution Files

You can go to the class web page and download a tar file containing all the files related with this project. Just access the link named Projects, and download the sources files associated with the link: *Project #2 – Movie Database with Lists.*

You implementation will consist of adding C++ code to implement two modules: SDoublyLinkedList.cc and DataManager.cc. You will receive all the .h files necessary for this project. In addition, you will be provided with a main program that uses the DataManager class, and interacts with the user to ask his/her input on the operations and polynomials to be evaluated. Finally, you will be given a Makefile with all the commands needed to compile and submit your project. In summary, you program will consist of the following files:

1. SDoublyLinkedList.h – interface for the sorted doubly linked list.
2. SDoublyLinkedList.cc – implementation of the sorted doubly linked list class container. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
3. list_test.cc – small test program for the sorted doubly linked list container.
4. list_test_exp.cc – template binding declarations for the the list_test.cc file. DO NOT MODIFY THIS FILE!!!!!!!!!!!!!!!!
5. Name.h – interface for an actor name class.
6. Name.cc – implementation of an actor name class.
7. NameList.h – interface for a list of actor names.
8. NameList.cc – implementation of the list of actor names.
9. Movie.h – interface for a movie information record class.
10. Movie.cc – implementation of a movie record.
11. DataManager.h – interface of the data management class for the movie database.
12. DataManager.cc – implementation of the data management class for the movie database. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
13. moviedb.h – interface for the movie database application.
14. moviedb.cc – implementation of the movie database application.
15. Makefile – file with the commandsto compile and submit you project.
16. test1.in – test input file 1.
    NOTE: YOU PROGRAM MUST PASS THIS FILE WITHOUT ERRORS IN ORDER TO BE CONSIDERED A RUNNING PROGRAM.
17. test1.out – expected output from test input file 1.
18. test2.in – test input file 2.
19. test2.out – expected output from test input file 2.

20. test3.in – test input file 3.
21. test3.out – expected output file from test input file 3.
22. prof_list_test – professor's version of the list_test program. NOTE: Known to be working correctly.
23. prof_moviedb – professor's version of the moviedb program. NOTE: Known to be working correctly.

PROJECT DUE DATE: **11:49 PM – November 20, 2001**.