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

ICOM 4035 – Data Structures
Fall 2002

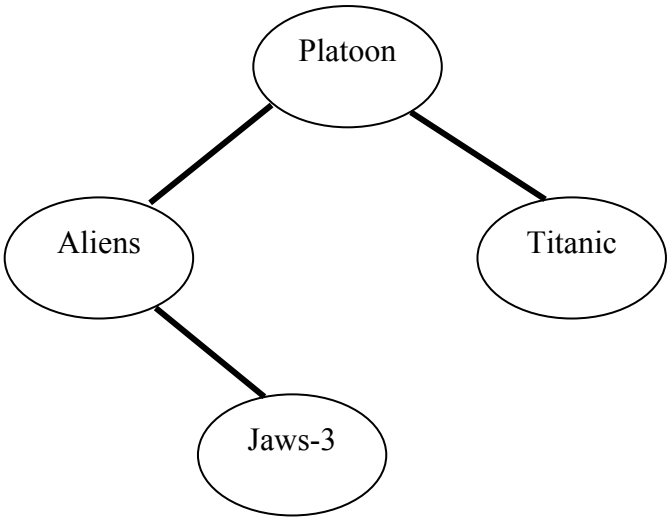
Project #3: Movie Database Application using Binary Search Trees
Due Date: 11:59 PM – December 4, 2002

Objectives

1. Understand the design, implementation and use of a binary search tree 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 iterators.

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 **Binary Search Tree** (BST) container class. The insertion of each movie record will be based on the movie title. For simplicity we will assume movie titles consisting of either one word, or multiple words separated by a dash - . For example: Platoon, Alien, Titanic, Jaws-3. The following diagram illustrates a general level organization of a movies database for these three movies, when inserted in that order



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 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 BST sorted order (in-order).

You must implement the DataManager class to store the movies in a Binary Search Tree (BST).

NOTE: You must keep the movies in the proper Binary Search Tree order at all times. Programs whose Binary Search Tree does not keep the BST order will be considered as not running. The BST order is a property defined as follows:

> **For any node *N* in the BST, the key of *N* is greater than the key of all nodes in its left subtree. Similarly, the key of *N* is less or equal to the key of all nodes in its right subtree.**

Binary Search Tree Container Class

You **must** implement a Binary Search Tree (BST) class container to keep the information about the movie records. The BST is a template that receives two parameter types:

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

The Binary Search Tree class container has two private data fields:

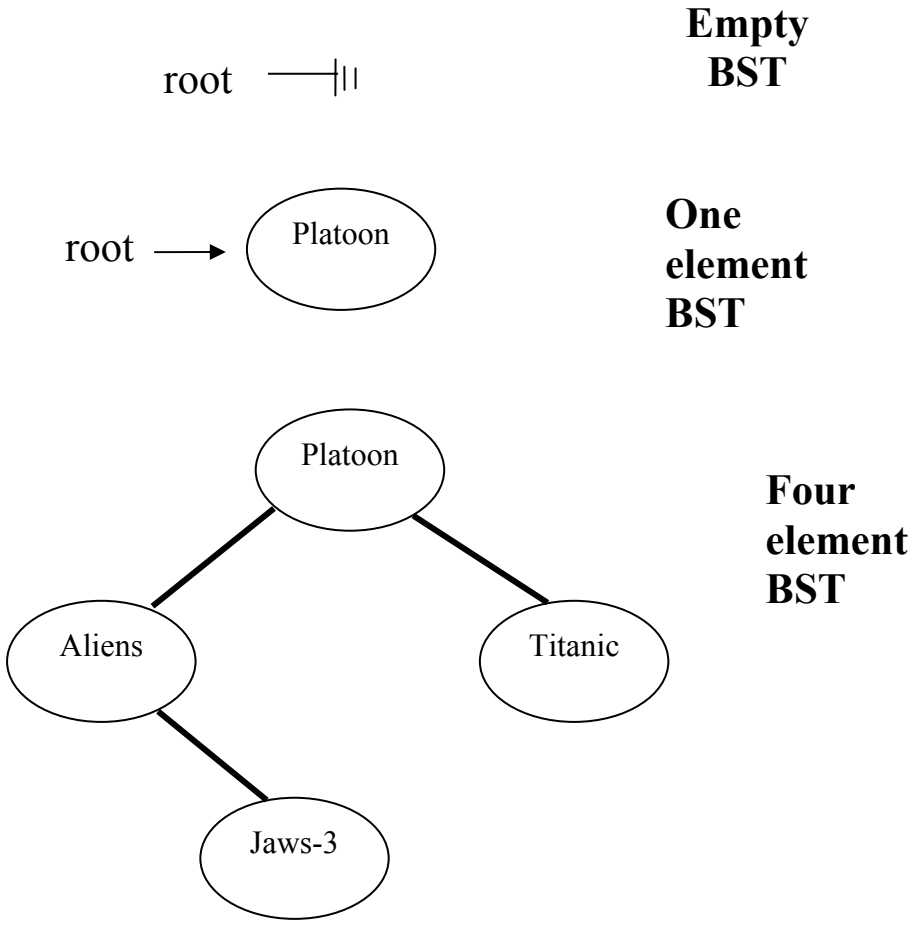
1. root– a pointer to the root of the Binary Search Tree (Recall that the root node stores data!!!).
2. t_size – the number of elements in the tree.

Each node *N* in the tree has three fields:

1. **data** - the data in each node is stored in this field.
2. **left_child** – pointer to the node that is the left child of the node *N*, and hence, the root of the left subtree of *N*.
3. **right_child** – pointer to the node that is the right child of the node , and hence, the root of the right subtree of *N*

When new nodes are inserted, they must be inserted so as to maintain the BST order. Likewise, when nodes are deleted from the BST, they must be removed so as to maintain the BST order.

The following diagram shows several cases of Binary Search Trees



NOTE: For Binary Search Tree 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 tree, skipping the node that you want to delete. Programs that deviate from this direction will not be considered a running program. The same applies to the usage of any other data structures (Linkedlist, vector, etc.)
2. The destructor cannot create memory leaks.
3. The copy constructor must create a deep copy of the tree.
4. The insert operation must put a new node in the appropriate order in the BST. If your tree does not follow the BST order property, your program will not be considered a running program.

Queue Container Class

You will need a queue to store tree node while executing in-order and pre-order tree traversals to create iterators. You will implement the queue container class using the toolkit cdlist_node for doubly linked lists. This time, however, the queue will not be circular. Instead, there will be a dummy node representing the front of the queue, and another dummy representing the rear of the queue. All insertions on the queue will be at the rear (in fact, before the rear node). All deletions will be at the front of the queue. Read the file queue.cpp for more details.

Sorted Circular Doubly Linked List Class Container

You will use your sorted circula doubly linked list class container from project 3, which was called List. This class was implemented using the cdlist_node toolkit, and the List container class, implemented in file List.cpp. This data structure will be used for two purposes:

1. Implement the list of actor names in a movie record.
2. Implement the list of movie records resulting from one of the operations to access the movie database, that is provided by the DataManager class.

You will need to add the following functions to your cdlist_node toolkit:

```
/*
 * This method inserts a new element after the node pointed to by
 * argument target pointer.
 * Parameter: obj - new object to be added into the list.
 * Parameter: target_ptr - pointer to a node, the new element will be added
 *            after it. This pointer is assumed not NULL.
 */
template <typename Item>
void list_insert_after(cdlist_node<Item> *target_ptr, const Item& obj);
```

```
/*
 * Copy a range of elements from one linked list into another linked list.
 * This is a deep copy operation. The functions receives two pointers
 * delimiting the nodes to be copied:  src_head_ptr and  src_tail_ptr.
 * The function will copy all element in the range:
 * src_head_ptr->get_next() and src_tail_ptr->get_prev().
 *
 * NOTE: The destination list is assumed to be empty!!! All pointers
 * are assumed not NULL.
 * Parameter: src_head_ptr - constant pointer to the node before the
 * first node of source list to be copied.
 * Parameter: src_tail_ptr - constant pointer to the node after the
 * last node of source list to be copied.
 * Parameter: dst_head_ptr - pointer to the head of the target list.
 */
template <typename Item>
void list_copy(cdlist_node<Item> *src_head_ptr,
               cdlist_node<Item> *src_tail_ptr,
               cdlist_node<Item> *dst_head_ptr);

/*
 * Removes all elements between two positions from the list.
 * The functions receives two pointers
 * delimiting the nodes to be removed:  src_head_ptr and  src_tail_ptr.
 * The function will remove all elements in the range:
 * src_head_ptr->get_next() and src_tail_ptr->get_prev().
 *
 * NOTE: The destination list is assumed to be empty!!! All pointers
 * are assumed not NULL.
 * Parameter: src_head_ptr - constant pointer to the node before the
 * first node of source list to be removed.
 * Parameter: src_tail_ptr - constant pointer to the node after the
 * last node of source list to be removed.
 */
template <typename Item>
void list_clear(cdlist_node<Item> *head_ptr,
                cdlist_node<Item> *tail_ptr);

/*
 * Counts all elements between two positions from the list.
 * The functions receives two pointers
 * delimiting the nodes to be count:  src_head_ptr and  src_tail_ptr.
 * The function will remove all elements in the range:
 * src_head_ptr->get_next() and src_tail_ptr->get_prev().
 *
 * Parameter: head_ptr - constant pointer to the node before the
 * first node of source list to be counted.
 * Parameter: tail_ptr - constant pointer to the node after the
 * last node of source list to be counted.
 */
template <typename Item>
std::size_t list_length(cdlist_node<Item> *head_ptr,
                        cdlist_node<Item> *tail_ptr);
```

NOTE: DataManager must keep the movie database in a Binary Search Tree (BST). The linked list is only used to throw in results from operations in the movie databases. Programs that use the linked list to store the movie databases will be considered as not running.

Name Class

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++ code 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++ code for this class.** (Except for the linked list code).

Data Manager

You must implement a Data Manager class that will maintain the Binary Search Tree of movies. The Data Manager has the BST 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. Read the file DataManager.cpp

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 #4 – Movie Database with Binary Search Trees*.

You implementation will consist of adding C++ code to implement four modules: cdlist_node.cpp, queue.cpp, BinarySearchTree.cpp and DataManager.cpp. In addition, you must supply your code for the SDoublyLinkedList class. 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. cdlist_node.h – interface for the circular doubly linked list.
2. cdlist_node.cpp – implementation of the circular doubly linked list class node. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
3. List.h – interface for the List container class.
4. List.cpp – implementation for the List container class as a sorted circular doubly linked list. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
5. list_test.cpp – small test program for the sorted doubly linked list container.
6. list_test_exp.cpp – template binding declarations for the the list_test_exp.cpp file. DO NOT MODIFY THIS FILE!!!!!!!!!!!!!!!!SDoublyLinkedList.h – interface for the sorted doubly linked list.
7. queue.h – interface of the queue class container.

8. queue.cpp – implementation of the queue class container. You will need this class to keep track of tree nodes during in-order and pre-order tree traversals. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
9. queue_test.cpp – test program for the queue container class.
10. queue_test_exp.cpp – template binding declarations for the queue_test.cpp.
11. BinarySearchTree.h – interface for the Binary Search Tree class container.
12. BinarySearchTree.cpp – implementation of the Binary Search Tree class container. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
13. test_bst.cpp – program to test your implementation of the BST.
14. test_bst_exp.cpp – template binding declarations for the the test_bst.cc file. DO NOT MODIFY THIS FILE!!!!!!!!!!!!!!!
15. Name.h – interface for an actor name class.
16. Name.cpp – implementation of an actor name class.
17. NameList.h – interface for a list of actor names.
18. NameList.cpp – implementation of the list of actor names.
19. Movie.h – interface for a movie information record class.
20. Movie.cpp – implementation of a movie record.
21. DataManager.h – interface of the data management class for the movie database.
22. DataManager.cpp – implementation of the data management class for the movie database. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
23. moviedb.h – interface for the movie database application.
24. moviedb.cpp – implementation of the movie database application.
25. explicit.cpp - template binding declarations for the moviedb.cpp file. DO NOT MODIFY THIS FILE!!!!!!!!!!!!!!!
26. Makefile – file with the commands to compile and submit you project.
27. test1.in – test input file 1.
    NOTE: YOU PROGRAM MUST PASS THIS FILE WITHOUT ERRORS IN ORDER TO BE CONSIDERED A RUNNING PROGRAM.
28. test1.out – expected output from test input file 1.
29. test2.in – test input file 2.
30. test2.out – expected output from test input file 2.
31. test3.in – test input file 3.
32. test3.out – expected output file from test input file 3.
33. prof_list_test – professor's version of the list_test program. NOTE: Known to be working correctly.
34. prof_queue_test – professr's version of the queue_test program. NOTE: Known to be working correctly.
35. prof_test_bst – professor's version of the test_bst program. NOTE: Known to be working correctly.
36. prof_moviedb – professor's version of the moviedb program. NOTE: Known to be working correctly.

Submitting Your Project

To submit your program, type the following commands

```
make sdir
make submit
```

PROJECT FIRM DUE DATE: **11:59 PM – December 4, 2002**.