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

ICOM 4035 – Data Structures
Spring 2003

Project #3: Event Management Application using Linked Lists
Due Date: 11:59 PM-April 10, 2003

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

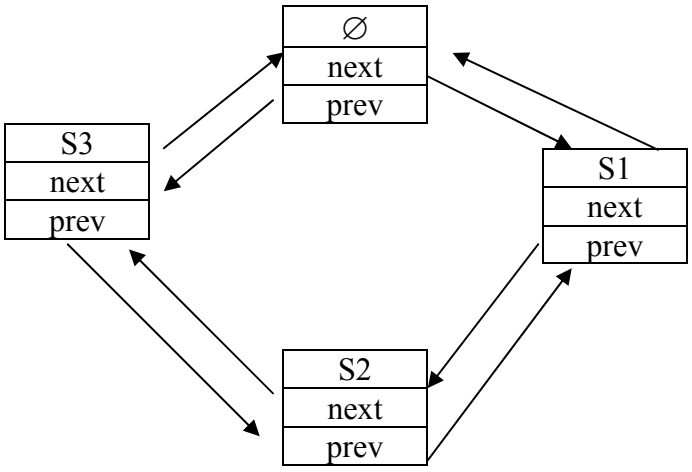
You will implement and test an application designed to manage events that are scheduled to be held on a group of conference room. In this application, a room is represented by a record that has the following three items of information:

1. Room number: a string indicating the unique identification number for a room.
2. Room capacity: an integer indicating the maximum number of persons that the room can accommodate.
3. Room Event List: A list of the events to be held in the room. The events are **ordered** by increasing chronological order.

The application will have one record per room available for scheduling of events. All room records will be kept in a list of rooms. In this list, the rooms will be **ordered** based on increasing room number. Thus, your application will manage an ordered list of rooms containing ordered lists of events (i.e. lists of lists!)

The list of rooms and the list of events will be implemented with a List container class, implemented as sorted circular doubly-linked list. As part of your task, you will develop all the functionality to implement a toolkit for sorted circular doubly-linked list, and then use this toolkit to implement the List container class. With this functionality in place, you will develop a scheduler class that will take care of adding new rooms, new events, searching for events, removing events, etc.

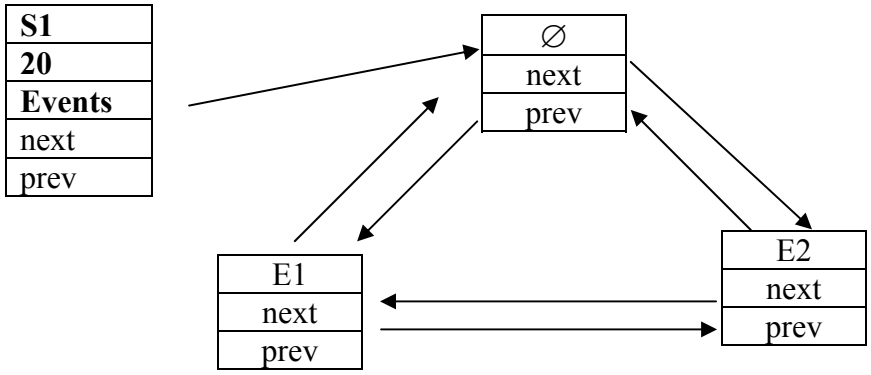
The following diagram illustrates a list with 3 rooms, numbered S1, S2, S3:



In reality, each node will not have simply the room number, but a **record** that has the following information:

1. Room number
2. Room capacity
3. List of events in a Room

The following diagram illustrates a record for room S2, where two events have been scheduled:



You will implement the following operations for the event management application, using a class called the Scheduler:

1. Add event – adds a new event to the system. The scheduler must find a room with enough capacity and without date conflicts to add the event.
2. Delete event – deletes an event from the system.
3. Find event by name – finds the information about a scheduled event based on the event name.
4. Find event by date – finds the information about scheduled events based on a date. Returns a new list with those events.

5. Find event by room – finds the information about scheduled events based on a room number. Returns a new list with those events.
6. Find event by time period – finds the information about scheduled event that will occur between two dates T1 and T2. Returns a new list with those events.
7. Remove all events in a room– clears all the events that were scheduled in a given room.
8. Add Room – adds a new room to the list of rooms available. The order in the list of rooms must be kept.
9. Print events in order – prints all events in chronological order starting from the first room.
10. Print events in reverse order – prints all events in chronological order, starting from the last room.
11. Exit the program

NOTE: You must keep the room in sorted order at all times, and you must keep the events in a room in order at all times.

Sorted Circular Doubly Linked List Class Container

You **must** implement a sorted circular doubly linked-list class container to keep the information about the room records, and also to keep the events in a room. To implement this class you will first develop a toolkit for implementing a node for circular doubly linked lists. Then, you will use this toolkit to implement a List container class that works as sorted circular doubly linked list. The circular sorted doubly linked list is a template that receives one parameter types:

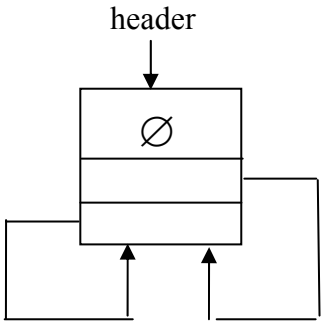
1. Item – the object to be stored in the list. It is assumed by the templates that a Item type will overload the relational operators = =, <, and >. In this project you will use classes Room and Event in this role of Item. These classes are explained below.

The sorted circular doubly linked list has one special node delimiting the actual nodes with the data in the list:

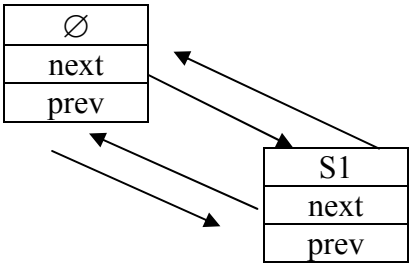
1. header node– has no data and points to the first and last elements in the list. If the list is empty, it points to the tail node.

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**.

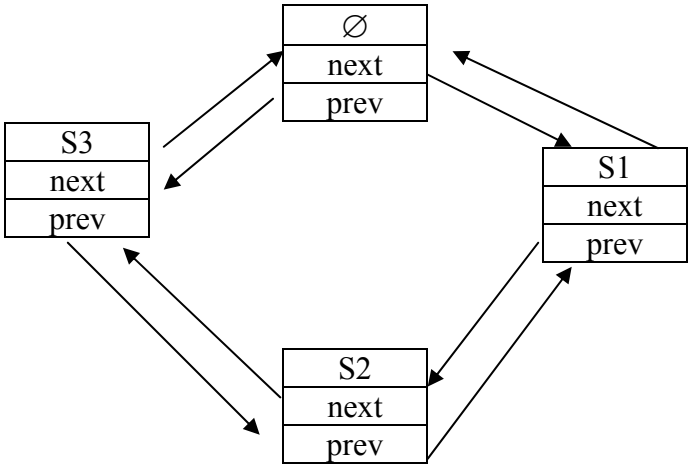
The following diagrams show several cases of sorted circular doubly linked lists:



Empty list



One element list



Three element list

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

1. **Implement the list of Room records in the system.**
2. **Implement the list of Events in a Room record.**

NOTE: For the circular 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.

Date Class

The Date class will represent the date of occurrence for an event. It consists of a year, month, day and hour (military time format). All these entities are integer private fields in the class. **You will be given the C++ for this class**.

Event Class

The Event class represents events that must be scheduled on the available rooms. Each event has a name, date, room number, and capacity. When a new event is added to the system, it does not have a room number. Your program will take care of finding a room for the event, and adding this information into the new event object being. **You will be given the C++ for this class, but you must implement the code that finds a room for a new event.**

Room Class

The Room class represents rooms in which events can be scheduled. Each room has a room number, room capacity and a list with the events scheduled for that room. Those events are kept sorted by chronological order. **YOU MUST IMPLEMENT THIS CLASS.**

Scheduler Class

You must implement a Scheduler class that will maintain the list of rooms available for scheduling events. The Scheduler has the list of rooms as private member. The Scheduler provides the interface to perform all the maintenance operations on the list of rooms. These operations were mentioned at the beginning of this document. **YOU MUST IMPLEMENT THIS CLASS.**

Event Manager

This is the main program for this application. It provides the interface for the user to add new events, view existing events, etc. **You will be given the C++ for this class.**

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 #3 – Event Management with Lists*.

You implementation will consist of adding C++ code to implement four modules: cdlist_node.cpp, List.cpp, Room.cpp and Scheduler.cpp. You will receive all the .h files necessary for this project. In addition, you will be provided with a main program that uses the Scheduler class, and interacts with the user to ask his/her input on the operations to be performed. 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.
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!!!!!!!!!!!!!!!!
7. Date.h – interface for the date class.
8. Date.cpp – implementation of the data class.
9. Event.h – interface of the event class.
10. Event.cpp – implementation of the event class.
11. Room.h – interface for a room information record class.
12. Room.cpp– implementation of a room record. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
13. Scheduler.h – interface of the event scheduler class.
14. Scheduler.cpp – implementation of the data management class for the movie database. YOU MUST IMPLEMENT THE METHODS THAT APPEAR IN THIS FILE.
15. EventManager.h – interface for the event management application.
16. EventManager.cpp – implementation of the event management application.
17. Makefile – file with the commands to compile and submit you project.
18. test1.in – test input file 1.
    NOTE: YOU PROGRAM MUST PASS THIS FILE WITHOUT ERRORS IN ORDER TO BE CONSIDERED A RUNNING PROGRAM.
19. test1.out – expected output from test input file 1.
20. test2.in – test input file 2.
21. test2.out – expected output from test input file 2.
22. test3.in – test input file 3.
23. test3.out – expected output file from test input file 3.
24. prof_list_test – professor's version of the list_test program. NOTE: Known to be working correctly.
25. prof_EventManager – professor's version of the moviedb program. NOTE: Known to be working correctly.

PROJECT DUE DATE: **11:59 PM – April 10, 2003**