Assignment:
For this project, you will implement a seminar database and search system. The idea is that organizers of training seminars will be able to post training seminar descriptions in the database, and users will be able to locate seminars based on keyword, location, time, etc. In some ways, this project is similar to your first project, except that you will use a 2-3+ Tree instead of a BST to provide access to the seminar records.

The 2-3+ Tree:
For information about the 2-3 Tree, read Section 10.4 in the textbook. However, the data structure you actually will implement is the 2-3+ Tree. A 2-3+ Tree is a modification to the 2-3 Tree that stores pointers to records in the leaf nodes, and stores key values only as direction finders in the internal nodes. Thus, internal nodes are quite different from leaf nodes in their structure. Section 10.5 in the textbook discusses B-trees in general (B-trees are the generalization of the 2-3 Tree) and the B+-tree in particular.

Seminar Records and Searches
A seminar record contains the following fields:

- **Title**: A string
- **Date/Time**: A string in the format YYMMDDHHmm where YY is the last two digits of the year, MM is the month, DD is the date, HH is the hour (24-hour clock) and mm is the minutes.
- **Length**: An integer (minutes).
- **Keywords**: A list of keywords. Keywords are strings.
- **Location**: Two unsigned short integers representing the X and Y coordinates (short integers are two bytes each).
- **Description**: A string.
- **Cost**: An integer (whole dollar amount).
- **ID**: An integer that uniquely identifies the seminar.

Users can search by a range of dates, by keyword, by location (within a given range) and by a range of costs. To support these possible search queries, you will index the database of seminar entries using one k-d tree and four 2-3+ Trees. The k-d tree will index records by location. One
of the 2-3+ Trees will index records by cost. One of the 2-3+ Trees will index records by date. One of the 2-3+ Trees will index records by keyword. One of the 2-3+ Trees will index records by ID. A record may contain multiple keywords. It will appear in the tree once for each keyword. Thus, if a given seminar has the three keywords “Database” “Woodworking” and “Accounting” then nodes with pointers to that seminar will appear three times in the keyword 2-3+ Tree, once for each keyword. Any given keyword search might (and typically will) generate multiple matching seminars.

**Input and Output**

There will be no input parameters to the program. Your program will read from standard input (**stdin**) and write to standard output (**stdout**). The input for this project will consist of a series of commands (some with associated parameters, separated by spaces), one command for each line. No command line will require more than 80 characters. Commands are free format in that an arbitrary number of additional spaces may be interspersed between parameters, and blank lines may appear between commands. You do not need to check for syntax errors in the command lines (although you do need to check for logical errors such as illegal duplicate insertions or deletions of records with non-existent keys).

Each input command should result in meaningful feedback in terms of an output message. Each input command should be echo’ed to the output. In addition, some indication of success or error should be reported. Some of the command specifications below indicate particular additional information that is to be output.

Commands and their syntax are as follows.

**insert**

```
insert ID
<date/time> <length> <x> <y> <cost>
<keyword list>
<description>
```

An insert command spans multiple lines. There will be no blank lines within an insert command. No line will require more than 80 characters. There can be multiple keywords on the keyword line, but there is only one line of keyword (thus, the keyword list is terminated by the newline symbol). Keywords may contain upper and lower case letters and underscores. The description itself can be an arbitrary number of lines long, and will be terminated by a blank line (which might contain spaces that should be ignored). Be sure to concatenate the description together, preserving the spacing and line breaks. It is an error to attempt to insert a record whose ID duplicates that of an existing record in the database. Such inserts should be ignored.

**search**

There are multiple forms of the search command, as follows:

```
search date <date/time> <date/time>
```

Print all records that fall within a range of date/times (inclusive).

```
search keyword <keyword>
```

Print all records that match the keyword.
search location \(<x>\) \(<y>\) \(<radius>\)
Print all records that fall within \(radius\) distance of the search point (inclusive).

search cost \(<low>\) \(<high>\)
Print all records that fall within the range (inclusive).

delete \(<ID>\)
Delete the record with the given \(ID\) from the database and all of the indexing trees.

dump date
dump keyword
dump location
dump cost
dump ID
Depending on the modifier to the dump command, print out a traversal of the corresponding tree. Use indentation and an appropriate traversal so as to be able to see the structure of the tree. For each seminar record, print out the relevant key (for the tree being dumped) and the seminar ID.

Implementation:
You should store only a single copy of each seminar record, with pointers to the record from the various tree indices. You should not use parent pointers in your 2-3+ Tree implementation. 2-3+ Tree nodes should be implemented with an abstract base node class and two subclasses, one for leaf nodes and one for internal nodes. You may use the STL linked list class for this project. You will probably want to store the keyword list for each record as a linked list.

Programming Standards:
You must conform to good programming/documentation standards, as described in the Elements of Programming Style. Some specifics:

- You must include a header comment, preceding main(). Your header comment should briefly describe what your program does.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Always use named constants or enumerated types instead of literal constants in the code.
- Precede every major block of your code with a comment explaining its purpose. You don’t have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation and blank lines to make control structures more readable.
- Precede each function and/or class method with a header comment describing what the function does, the logical significance of each parameter (if any), and pre- and post-conditions.
• Decompose your design logically, identifying which components should be objects and what operations should be encapsulated for each.

Neither the GTAs nor the instructors will help any student debug an implementation unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or code taken from the textbook. Note that the textbook code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point. You may not use code from STL, MFC, or a similar library in your program except where noted above (i.e., you may use the STL linked list class).

Testing:

A sample data file will be posted to the website to help you test your program. This is not the data file that will be used in grading your program. The test data provided to you will attempt to exercise the various syntactic elements of the command specifications. It makes no effort to be comprehensive in terms of testing the data structures required by the program. Thus, while the test data provided should be useful, you should also do testing on your own test data to ensure that your program works correctly.

Deliverables:

When structuring the source files of your project (be it in Eclipse as a “Managed C++ Project,” or in another environment), use a flat directory structure; that is, your source files will all be contained in the project root. Any subdirectories in the project will be ignored. If you used a makefile to compile your code, or otherwise did something that won’t automatically compile in Eclipse, be sure to include any necessary files or instructions so that the TAs can compile it.

If submitting through Eclipse, the format of the submitted archive will be managed for you. If you choose not to develop in Eclipse, you will submit either a ZIP-compressed archive (compatible with Windows ZIP tools or the Unix `zip` command) or else a tar’ed and gzip’ed archive. Either way, your archive should contain all of the source code for the project, along with any files or instructions necessary to compile the code. If you need to explain any pertinent information to aid the TA in the grading of your project, you may include an optional “readme” file in your submitted archive.

You will submit your project through the automated Web-CAT server. Links to the Web-CAT client and instructions for those students who are not developing in Eclipse are posted at the CS2606 website. If you make multiple submissions, only your last submission will be evaluated. Note that only one member of the pair will make a submission. Whatever is the final submission from either of the pair members is what we will grade.

In addition to the project submission, you are also required to submit a schedule and weekly updates to that schedule. You will find the schedule template at the course website. The initial schedule is due Friday, September 22 by 11pm. You must submit updates on Friday, September 29 by 11pm and Friday, October 6 by 11pm. You must also submit (as part of your final submission package) a final schedule with the “elapsed” column filled in indicating the total hours that you spent to complete all aspects of the project. You won’t receive direct credit for submitting the schedule as required, but each instance of failing to submit scheduling information as required will lose five points for you and your partner from the project grade.
Finally, you and your partner are required to spend a 2-hour block of time in the McBryde 116/118 labs during the TAs posted office hours for each designated “lab week” During this time, you are expected to work together in “pair programming” mode, where one of you is “driving” at the keyboard, typing, while the other is “navigating”. There are two designated lab weeks for this project, 9/25-9/29 and 10/2-10/6. While you do not directly receive points for attending lab week sessions, each failure to do so will lose you and your partner five points from the project grade.