Project 3

Programming and Algorithms II CSCI 211

This is an individual assignment, do your own work!

Objectives

- Understand Abstract Data Types (be able to define and implement C++ classes).
- Practice information hiding (encapsulation, in other words, making the internals of your classes private).
- Design a Linked List.
- Review dynamic memory allocation and deallocation.
- Gain more experience with pointers.

Overview

Implement a sorted linked-list of nodes containing pointers to Video objects. Use the video titles to keep the list sorted.

Implement a main() function that reads and executes commands read from stdin. Your program must handle the following commands:

- Create and insert a new video into the linked-list
- Print all the videos in the list.
- Lookup a video by title and print it.
- Output the number of videos in the list.
- Remove a video from the list, by title

Tip: Start early and work daily.

<continued on next page…>

Program Requirements

Implementation

Use the Video class you created for project 2 to store videos.

Create a linked-list class named Vlist. Put this class in the files vlist.h and vlist.cpp. The starter code for vlist.h is provided in the project 4 starter pack.

Your Vlist of Video objects must always be ordered alphabetically by video title, that is, the Vlist "insert" method must insert the new Video object passed-in so that after the insert all videos in the list are ordered alphabetically.

All member variables of class Video and class Vlist must be private. Use heap allocation to create and delete Video and Node objects. When creating a new Video object, you must use the new operator (just like in Project 2).

Create each Video object in your main function, and then add it to your Vlist object.

When inserting into the Vlist, you must use the new operator to create a new Node object (just like in class linked-list examples).

Create each "new" Node object in the Vlist::insert() method. When removing a video from the list, both the Node and the Video object must be deleted.

You must provide a correct destructor for class Vlist. It must delete all Video and Node objects.

There must not be any input or output in class Vlist (you cannot use cin/cout/cerr in a Vlist member function). However, you can call Video's print function.

The only input or output allowed in class Video is output in Video::print() function.

Return an exit status of 0 if all the input commands (defined below) were legal. Return an exit status of 1 if any of the commands were illegal.

<continued on next page…>

Input

Your program must read commands until the end of input (until user presses ˆD or the end of a redirected file is reached. When end of input is reached, exit your program with an exit status of 0 (see below for exit status when a bad command is entered).

Your program must handle the following commands. supplied on subsequent input lines.

Some commands are followed by additional lines of data that you must read:

Command Name	**Command Action**	**Command Data**	**Error Conditions**
insert	Insert a new video into the linked list in **sorted** order, according to video title. (This is similar to the "insertAfter()" method we did in class. Traverse nodes and stop at the node **before** the spot for the new node).	title, url, comment, length, rating	Title is already in the list. See error messages below
print	Print each of the videos in the list using the same output format from Project 2.	none	none (print nothing if list is empty)
length	Write the number of videos in the list as a single integer to stdout (don't output anything other than the integer).	none	none (output 0 if the list is empty)
lookup	If the given title is in the list, print that video using the format from Project 2.	title (may contain spaces)	Title is not in list.
remove	If the given title is in the list, remove it.	title (may contain spaces)	Title is not in list.

<continued on next page...>

Errors and Error Messages

The input will not contain any errors other than those listed in this section. For example, the insert command will always be followed by a title, url, comment, length, rating (each on a separate line). The program should NOT terminate for the following three errors:

Command	**Error**	**Stderr Error Message Output****
insert	Title already in list.	Could not insert video <XXX>, already in list.
lookup	Title not in list.	Title <XXX> not in list.
remove	Title not in list.	Title <XXX> not in list, could not delete.

**Replace XXX with the title entered by user, print the < and >.

If the user enters a command other than (insert, print, length, lookup, remove) print the following error message and terminate the program with an exit status of 1. Replace the XXX with the command entered (make sure you print < and > in your error message).

<XXX> is not a legal command, giving up.

Example Execution

Use the tests provided in the starter pack to see the expected output for various inputs provided.

<continued on next page...>

Tips & Tricks

1. Program incrementally. Get small parts working before you move on.
2. “cin >>” will not work for strings with spaces. You must use getline() to read input that might contain spaces. For example, the ‘lookup’ and ‘remove’ commands provide titles for their command data. You must use getline() to read these titles.
3. The linked list examples from class are all linked lists of integers. For this assignment, each node’s value is a pointer to a Video object instead of just a simple integer.
4. Use stack memory in main() to create the Vlist object.
5. Use getline() to read all the commands. getline() will automatically throw away the newline character at the end of the line.

<continued on next page…>

Plan of Attack

Before you start coding, have a look at the provided input files (e.g. tests/t01.in and so on). This will give you a good feel for how the input is arranged. Make sure you understand how the input relates to the commands described above.

Step 1: Copy Project 2’s main.cpp, video.h, and video.cpp to your p3 directory.

Step 2: Create new file vlist.cpp (your Videos list class). A starting version of vist.h is provided in the starter pack for this project.

You will add additional methods to the Vlist class as you go along, but this should get you started in the right direction.

Step 3: Update main() so it includes vlist.h. You may be able to repurpose the project 2 code that reads the video records.

Step 5: Compile your files using **make** and fix any compile warnings and errors.

Step 6: Write the code to input and process each of the commands specified in the **Input** section above. It is recommended that you implement each command in the order given in the table. Test as you go along. For example, implement **insert** and **print**, then use **print** to make sure **insert** is working correctly.

Step 8: Write the destructors for Vlist and Node.

Step 9: Review the requirements carefully to make sure your program is complete before submitting.

General Requirements

The first lines of all your files (both .h and .cpp) must contain the following comments:
// filename
// Your first and last name
// Your Blackboard user id

There must be a brief descriptive comment above each function or method implementation.

There must be in-line comments within the body of your code, at least one for every several lines of code or each code block.

Format your code. Make sure your code blocks align. There is no required indentation standard, but be consistent throughout your program.

Use descriptive variable names. Avoid ambiguous or short variable names.

Use a capital letter as the first letter for classes in your code (e.g., Video). Use all lower case for file names (e.g., video.h).

Keep the main() function relatively short. No more than 30 lines max. Split your main function into smaller sub functions as needed.

Testing

Use **run_tests** to test locally. Do not submit to TurnIn until all of the local tests are passing.

Submission

When all of the local tests are passing, and you are you you have met all of the requirements stated above, submit your source files (**video.h video.cpp vlist.h vlist.cpp main.cpp**) to Turn-in.