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.
- More dynamic memory allocation and deallocation.
- Gain more experience with pointers.

Overview

Implement a sorted linked-list of nodes containing pointers to Video objects. Keep the list sorted by increasing order of video title (alphabetically).

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

- **insert:** Create and insert a new video into the linked-list
- **print:** Print all the videos in the list.
- **lookup**: Find a video by title, and if found, print it.
- **length:** Output the number of videos in the list.
- **remove:** Remove a video from the list, by title

<continued on next page...>

Input

Your program must read commands until the end of input. When end of input is reached, exit your program with an exit status of 0.

Your program must handle the following commands.

Some commands are followed by 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.	title, url, comment, length, rating	Title is already in the list. See error messages, on next page.
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 (use the **Video::print** method).	title (may contain spaces)	Title is not in list. See error messages, on next page.
remove	If the given title is in the list, remove it.	title (may contain spaces)	Title is not in list. See error messages, on next page.

<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 <*title*>, already in list.
lookup	Title not in list.	Title <*title*> not in list.
remove	Title not in list.	Title <*title*> not in list, could not delete.

**Replace *title* with the title read from the input, DO output the < and >.

General Requirements

- No more than 30 lines in your main function.
- Use new operator to create the Node and Video objects.
- All heap memory is deallocated prior to program exit.
- Do not use cin, cout or cerr within your Vlist class implementation.
- Code properly aligned (proper use of indents)
- Header Comment in all .cpp and .h files (last name, first name, ecst_username)
- Comment above each function
- Inline comments within function bodies.
- No more than three submissions to turn-in.

...continued on next page...

Steps

1. Before you start coding, have a look at the provided input files (e.g. tests/t01.in). 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. Examine the corresponding output files to make sure you understand the expected output (e.g. tests/t01.out).
2. Copy Project 2's video.h, and video.cpp to your p3 directory.
3. Create new file vlist.cpp (your Videos list class), with the following content:
   ```
   #include "vlist.h"
   using namespace std;
   ```

4. A starting version of **vlist.h** is provided in the starter pack for this project. You will add and implement new methods for class **Vlist** as you go along. For example, when you implement the **insert** command in **main.cpp**, you will need to define and implement a public **insert** method in **vlist.h** and **vlist.cpp.** You will likely find the example code under **Lecture Material > Lecture Notes > Weeks 3 & 4 > Code Examples** very helpful, as the **SortedList** class we implemented in class is very similar to what you'll need for the **Vlist** class, except that the **Node** class's data member variable **m_value** type will be **Video*** instead of **int.**

5. Create file main.cpp with the following content:

   ```
   #include <iostream>
   using namespace std;
   #include "video.h"
   #include "vlist.h"

   int main() {
      return 0;
   }
   ```

6. Create a Vlist object named vlist on the memory stack, at the top of your main function.
7. Create the following function above **main**:
   ```
   void read_input(Vlist &vlist) {
   }
   ```
8. Call **read_input** from **main**, passing in the **vlist** object. Note the use of "pass by reference" here. In this case, we want to pass a pointer to the **vlist** object we created on the memory stack and not a copy of the object, but you can still use the non-pointer syntax to use the vlist object within the **read_input** function.
9. Implement the **read_input** function to process each command as it is read from stdin, using a **while** loop. Use the **getline** function to read each command into a string variable named **command**.
   ...continued on next page...

10. Inside the **while** loop, after each new command is read in, use a sequence of **if/else** statements to handle each of the valid command strings (**insert**, **print**, **length**, **lookup**, or **remove)**. Some commands, like **insert**, will require reading additional input from **stdin**. See the table on page 2 for the details for handling each command. I recommend implementing and testing each command in sequence, one at a time, in the command order listed above. Each of the commands requires a new public method to be added to class **Vlist**. For example, you might define the public **insert** method like this, inside **vlist.h**:
    void insert(Video *video);
    ...and then implement this method within **vlist.cpp**, and call the new method from **main.cpp** when handling the **insert** command.
11. After you have implemented the code to handle all of the valid commands, add one additional **else** clause at the end to handle the case of an invalid command (this else clause will only execute if the command string didn't match any of the valid command strings). Inside this final **else** clause, write the following error message to **stderr:**
    <*command*> is not a legal command, giving up.
    Substitute the invalid command string for ***command.***
    Note: DO output the < and > characters as shown above.
    Then terminate the program with an exit status of 1, either by calling **exit(1),** or by returning the value 1 from main. Since your input code is in a separate function, it will likely be easier to call **exit(1)** from inside the **read_input** function.
12. Implement the destructors for classes **Vlist** and **Node** such that all heap allocated memory is deallocated when the **Vlist** object you allocated at the top of **main** goes out of scope.
13. Review the requirements carefully to make sure your program is complete before submitting.

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.