Project 2

Programming and Algorithms II CSCI 211

Note: Complete Lab 3 before starting Project 2 (you will implement class Video in Lab 3).

Read these instructions completely and thoroughly. Understand the task and plan your approach before beginning coding.

Objectives

- Practice with classes and objects
- Read input from "Standard In" using both white-space delimited (token) input **(>>** operator) and line-based input (calling the **getline** c-library function).
- Comparison of strings
- Dynamic memory allocation
- Writing output to "Standard Out" and "Standard Error"
- Program exit codes (the integer value returned by the main function).
- Introduce sorting algorithms

Overview

Read an unordered collection of video records, each containing a **title**, **URL**, **comment**, **length**, and **rating**. Create a Video object for each video record and store the video object pointers in an array.

Sort the video object pointers within the array, by **rating**, **length** or **title** (as specified in the input).

Output all the video fields to Standard Out in sorted order.

Tip: Start early and make steady daily progress.
...continued on next page...

Program Input

The input begins with the sort method, specified on the first line:
sort-method (**rating**, **length** or **title**)

rating: sort the output in decreasing numerical order of rating, highest to lowest.
length: sort the output in increasing numerical order of length, lowest to highest.
title: sort the output in increasing alphabetical order of title, lowest to highest.

After the sort criteria line is a sequence of one or more sets of video records. Each video record consists of five separate lines of input, as follows:

Title (A string)
URL (A string)
Comment (A string)
Length (In hours, floating point)
Rating (Integer: 1, 2, 3, 4 or 5)

Here is an example input set consisting of the ***sort-method*** followed by two video records:

```
title
United Break Guitars
http://www.youtube.com/watch?v=5YGc4zOqozo
Great example of one person getting a giant company to listen
4.5
3
Spaces Versus Tabs
https://www.youtube.com/watch?v=SsoOG6ZeyUI
Decide for yourself: spaces or tabs?
2.83
4
```

...continued on next page...

Program Output

Consider the following input:

```
rating
United Break Guitars
http://www.youtube.com/watch?v=5YGc4zOqozo
Great example of one person getting a giant company to listen
4.5
3
Spaces Versus Tabs
https://www.youtube.com/watch?v=SsoOG6ZeyUI
Decide for yourself: spaces or tabs?
2.83
4
It’s Not About the Nail
https://www.youtube.com/watch?v=-4EDhdAHrOg
Favorite web video
1.68
5
Pet Interviews - Guinea Pig
https://www.youtube.com/watch?v=jW3XtKBlTz0
Best guinea pig interview
1.75
1
```

The following is the correct output for the above input (sorted by decreasing order of rating as specified in the above input):

```
It’s Not About the Nail, https://www.youtube.com/watch?v=-4EDhdAHrOg, Favorite web video, 1.68, *****
Spaces Versus Tabs, https://www.youtube.com/watch?v=SsoOG6ZeyUI, Decide for yourself: spaces or tabs?, 2.83, ****
United Break Guitars, http://www.youtube.com/watch?v=5YGc4zOqozo, Great example of one person getting a giant company to listen, 4.5, ***
Pet Interviews - Guinea Pig, https://www.youtube.com/watch?v=jW3XtKBlTz0, Best guinea pig interview, 1.75, *
```

Note that the rating number (1 to 5) is output as a sequence of asterisk (*) characters, just like you did in Lab 3.

...continued on next page…

Steps

1. Complete Lab 3 now if you haven't done so already.

2. Copy files video.cpp and video.h from **~/csci211/labs/lab3/** into your **~/csci211/projects/p2/** directory.

3. Create file **main.cpp** with the following starting content:

   ```
   #include <iostream>
   using namespace std;

   #include "video.h"

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

   Note: We use #include <> to include built-in C++ library interfaces, and we use #include "" to include the .h files that we creates.

4. Use the provided **Makefile** to build your Project 2 program, **videos**. To do this, simply type the make command, which will read the provided **Makefile** and attempt to build the **videos** executable:

   ```
   make
   ```

   Verify that the program is created without any errors or warnings. Fix any errors and warnings. After running **make,** always stop to fix any warnings or errors. This habit will save you time in the long run.

5. Edit file **main.cpp**, declare a constant named **MAX** to define the maximum allowed number of video records to process (100), declare your array of video object pointers, and initialize each pointer in the array to the value NULL:

   ```
   #include <iostream>
   using namespace std;

   #include "video.h"

   const int MAX = 100;

   int main() {
      int result = 0;
      // Declare and initialize an array of pointers to Video objects, on the runtime memory stack.
      Video* videos[MAX] = {NULL};
      return result;
   }
   ```

   ...continued on next page...

6. Declare a new **string** variable in main named **sort.** Read the first line of input into the **sort** variable using the **getline** function. The first parameter to **getline** is the **input stream** to read from, in this case **cin**, the second parameter is a **string** variable to receive the value read from the input.
   Validate that the sort string read in is valid, that is, it must exacly match either "rating", "length" or "title". If the sort string read in is not valid, write the invalid sort method string to stderr followed by the string " is not a legal sorting method, giving up." followed by a newline, then return 1 from main (exit the program immediately.
7. Declare a set of local variables near the top of main, to store the fields for each video record (**title**, **url**, **comment**, **length, rating**). You need to determine the appropriate type for each variable.

8. Declare an integer variable named **video_count** and initialize it to 0. You will use this variable to keep track of how many video records you have read in, and what the next available **videos** array index is.

9. Create a **while** loop to read the video records, one at a time, until the end of input (EOF) is reached, like this:
   ```
   while(getline(cin, title)) {
    getline(cin, url);
    // ...add code here to read remaining video record fields.
    // Read comment and url with getline
    // Read length and rating via cin
    // Don't forget cin.ignore(); at the bottom of the while loop
   }
   ```
   The first call to **getline** (in the while expression) tries to read the next video record title, if one is available. If successful, **getline** returns true, the new value is copied into the **title** variable and the while loop executes, otherwise, when **EOF** is reached (there is no more input to read), **getline** returns false and the while loop exits. If the title exists, you may assume the rest of the video fields exist.
10. The input is allowed to contain up to 100 video records. If more than 100 video records are specified, print the message “Too many videos, giving up.” and a newline (endl) to “Standard Error” (cerr), then immediately terminate the program by returning the value 1 from main. Hint: Use variable **video_count** to know when the max has been exceeded. This check should be done inside the while loop before attempting to read in any more input.
11. Once inside the while loop, you may assume that all of the remaining video record fields (**url**, **comment**, **length**, **rating**) will be present in the input (there are no incomplete Video records in the input).
    Note: In the input, **title, url,** and **comment** are strings that contain whitespace, one string per line, and so **getline** is used to read each ot them**,** but the other two video record fields, **length** and **rating** are numeric fields. **getline** can only be used to read string values from the input, one line at a time. Numeric values must be read in using the **>>** (token based) input operator, like this:
    cin >> length;
    The **>>** operator is able to convert the character values read from the input stream into the appropriate varable type as needed, in this case, converting a sequence of characters like '5.25' from the input into the floating point numeric value 5.25.
12. Important Note: AFTER READING FROM **cin** using the **>>** operator, YOU MUST CALL **cin.ignore()** before calling **getline()** again, otherwise any whitespace left on the current line of input after the last token read by **cin** will be consumed as the next line of input by **getline**. The best way to handle this is to simply make sure the last line inside your while loop is:
    cin.ignore();
    This ensures that the next call to **getline**, in the while loop test expression, will work properly.
    Important note: You should only call **cin.ignore()** BETWEEN using the **>>** input operator and calling **getline**. You should not call **cin.ignore()** after calling **getline.** Adding unneeded calls to **cin.ignore()** can cause problems (**cin.ignore()** always consumes at least one character from **cin,** even if the next character isn't whitespace...unfortunately this can cause bugs that are difficult to trace so be careful using **cin.ignore()**. Only use it **BETWEEN** using **>>** and **getline**).

13. In your while loop, just after you read each video record, create a new **Video** object (using the **new** operator). Pass each of the varaibles you just read in to the contructor as input parameters.
14. Insert the new VIdeo object you just created into your **videos** array, at the next available array index (using variable **video_count**). ...continued on next page...
15. Increment (add 1 to) **video_count.**
16. After the input **while** loop, write a **for** loop that calls the **print** method on each video object pointer stored in the **videos** array. Use the **video_count** variable (which contains the total number of video objects created) to know when to stop the loop).
17. Use **make** to rebuild the **videos** program from the latest code, fix any warnings and errors, and then manually run test case #1 to validate that your input and output code is working correctly so far.
    ./videos < tests/t01.in
18. The exected output is the video records in the order they appear in the input. You can view the input for test case 1 using this command:
    cat tests/t01.in

19. After the while loop, implement the code to sort all of the video object pointers according to the specified **sort-method** read from the first line of input. To sort the Video objects by **rating**, **length** or **title**, you first need to define (in videos.h) and implement in (videos.cpp) the following comparison methods in class Video.

```
// Return true if otherVideo's rating is greater than this Video's rating.
bool compareRatings(Video *otherVideo);

// Return true if otherVideo's length is less than this Video's length.
bool compareLengthsVideo *otherVideo);

// Return true if otherVideos's title is less than
// (sorts alphabetically before) this Video's title.
bool compareTitles(Video *otherVideo);

// Return true if otherVideo should be sorted before this Video, according to
// the given sort string, which can be either "rating", "length", or "title".
bool compare(Video* otherVideo, string sort);
```

Next, you need to apply the "Bubble Sort" algorithm to rearrange the **Video** object pointers in the **videos** array according to the specified sort criteria. You may use the following code as-is:

```
// Compare and swap adjacent pairs of Video object pointers until they
// have been reordered according to the user provided sort criteria.
for (int last = video_count - 1; last > 0; last--) {
      for (int cur = 0; cur < last; cur ++) {
            // Swap the object pointers in the array.
            if (videos[cur]->compare(videos[cur+1], sort)) {
                  swap(videos[cur], videos[cur+1]);
            }
      }
}
```

Note: To avoid having to copy and paste the code above from these PDF instructions (which is never a good idea), the same code shown above is provided in the Blackboard Project 2 items.

20. Write all the **Videos** to "Standard Out" in sorted order by writing a **for** loop to call the **print** method on each **Video** object pointer stored in the **videos** array.
21. Free each **Video** object pointer stored in the **videos** array.
22. Return the appropriate integer value from **main**. ...continued on next page...

Test

Since this program requires error messages be written to “Standard Error” (cerr), in addition to output written to Standard Output (cout), the test cases include standard output (e.g. t01.out) and error output (e.g. t01.err) files.

Use the **run_tests** script to test **videos**.

Submission

When everything is complete, working as expected and passing all of the provided tests, have another look at the grading criteria for the project (under Projects > Project 2) and make any needed changes to your source code.

When everything is complete, submit your program files to https://turnin.ecst.csuchico.edu: video.h video.cpp, main.cpp.

Reminder: Submit only your C++ files (.h, .cpp). Do not submit your object files or your executable program file.

If you do not finish in time, turn in whatever work you have before the final cut-off. If you turn in nothing, you get a zero. If you turn in something, you may at least receive partial credit.