Project 2

Programming and Algorithms II CSCI 211

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 input (cin) and line-based input (getline()).
- 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 descriptions, each containing a title, URL, comment, length, and rating.

Allow the user to specify how the videos should be sorted: by rating, by length, or by title.

After all the descriptions have been read, print them to standard output (using cout) sorted using the specified criteria (rating, length, or title).

Tip: Start early and make steady daily progress.

Program Requirements

Create a class to store the video descriptions. Call this class Video and put it in the files video.h and video.cpp. All the member variables in class Video must be private.

Put the main() function in the file main.cpp

Allow the user to specify up to 100 videos. If more than 100 videos are specified, print the message "Too many videos, giving up." and a newline (endl) to "Standard Error" (using cerr) and terminate the program.

Return an exit status of 0 (from your main function) if there are no errors, 1 if there are any error conditions detected.

Use an array of pointers to videos (Video*) to store the videos. Use dynamic memory (the "new" operator) to create a new Video object for each description read. Don't forget to use the "delete" operator to free each dynamic allocation before the program exits.

Note: 100 is the maximum number of input records. You can statically allocate an array of 100 elements.

The input begins with the sort method, specified on the first line:
<sort method> ('rating', 'length' or 'title')

rating: sort the output so the highest ratings come first.
length: sort the output so the shortest videos come first.
title: sort the output so the videos titles are in alphabetical order.

If the first line is not one of the above strings, print the message "XXX is not a legal sorting method, giving up." to "Standard Error", where XXX is the illegal sorting method. Be sure to include a newline (endl) character at the end of the message. Then terminate the program with a non-zero error status (return 1 from your main() function).

After the sort criteria is a sequence of one or more video description 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 Code> (Integer: 1, 2, 3, 4 or 5)

Here is an example input set consisting of the **sort method** followed by one **video record**:

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

If two or more Videos have the same values for the current sorting method (e.g. you are sorting by length and two or more videos have the same length), these videos should be sorted in the same order as the input. For example, in the sample input below if both Dog Playing Piano and Cupcake eating challenge had the same rating, Dog Playing Piano would be first because it is first in the input.

Continued on next page…

Example Execution

Consider this 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 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, *

Tips

1. Make sure you understand how the input is ordered and the required output before you start.
2. Program incrementally. Get small parts working before you move on. The next section describes how to break the program into pieces.
3. cin >> my string will not work for strings with spaces. You will need to use getline(cin, title), where title is a C++ string variable.
4. Use the call to getline in the while loop:

```
// while there are more video descriptions to read
while (getline(cin, title)) …
```

5. You may use “cin >> ” to read the length and the rating fields.
6. After you read the rating (which is the last element of each video description) there will be an extra “end of line” character that you need to get rid of before you can read the next video description. Use the following statement to get rid of that extra “end of line” character: cin.ignore();
7. When sorting the videos you need to be able to determine how two video objects should be ordered. The easiest way to do this is to write member functions to handle the comparisons in class Video. For example, this method could be used when sorting the videos by length:

```
// return true if the current video is longer than the given video (other) ,
// otherwise return false
bool Video :: longer(Video *other) {
    return (mlength > other -> mlength ;
}
```

8. You will need to compare strings alphabetically when sorting by title. If you use C++ style strings (as suggested above) you can compare them using the normal comparison operators:

```
string str1 = "hello";
string str2 = "goodbye";
if ( str1 > str2 ) {
    // this code is executed if str1 is alphanumerically greater than str2
}
```

9. Use the bubble sort algorithm to sort the videos (more info at https://en.wikipedia.org/wiki/Bubble_sort). The following code uses the Video::longer() function above and sorts the array by video length:

```
for (int last = num_videos - 1; last > 0; last--) {
        for (int cur = 0; cur < last; cur ++) {
                // Swap the object pointers in the array.
                if (videos[cur]->longer(videos[cur+1])) {
                        swap(videos[cur], videos[cur+1]);
                }
        }
}
```

Suggested Plan of Attack

Note: You probably have already done Steps 1 & 2 below as a part of Lab 3, if so, you may copy files video.cpp and video.h over into your project 1 directory, and skip to Step 3 below.

Step 1: Create video.h and put an empty class definition and the usual #ifndef / #endif preprocessor macros (to prevent the same header file from being included more than once by any cpp file).

Step 2: Create an empty video.cpp that includes video.h.

Step 3: Create main.cpp and write an empty main() (a main function that does not do anything) main.cpp should also include video.h.

Step 4: Use the provided Makefile to build your Project 2 program, videos.

Step 5: Write the code in main() to read the input (the sorting method string and the video descriptions). Make sure you continue reading video descriptions until the end of input.

Tip: cin is an instance of class istream, which contains many useful methods. For example, you have use cin.ignore and getline() in this project. For your while loop condition, you should use the cin.peek() method. You may read the related documentation here.

Step 6: Implement the Video class. Include a constructor and a print function.

Step 7: After you read each description, create a new Video object (using the new operator). Call the print method on that new object. Test to make sure the output is in the correct format. Here is some example code to get you started:

```
// Create the new Video object
Video *videoObj = new Video(??? );

// Replace ??? above, with the variables that hold the values you read in
// Output the content of this Video object, just to validate that everything is
// working properly so far (you will remove this call to print() later.
videoObj->print();
```

Step 8: Use an array of 100 pointers to Video objects to store the videos. Instead of printing after each one you read, print all the Videos in the array after you are done reading all the input. The output of your program is now complete except for the order of the videos. Make sure everything is working before you move on.

Step 9: Implement the sorting of the videos. Do not start this step until you have tested your program and are sure all the other components are working.

Compile & 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 provided run_tests script to test Project 2. If you haven’t yet setup run_tests, now is the time to do so. See Guides > Advanced Local Testing for info on how to do this.

Tip: You can view the test data within the tests directory provided in the project 2 starter pack.

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 Course Content > Projects > Project 2) and make any needed changes.

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.

Each student will complete and submit this assignment individually. Watch the due date. Work on this project every day until it is completed.

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.