Program #1 - Video List

Prof Bill - Jan 2017

Program #1 logistics:

- Emphasis: linked lists, pointers, learning a new programming language
- Due: **Wed Jan 18, 2017** at the beginning of class
- Worth: **8 points** (8% of your grade)

1. Description

William Tee owns Ye Olde Video Shoppe, the last VHS Video Cassette rental store in the tri-county area. He currently writes down customer video preferences in a spiral notebook that sits beside the cash register.

The good news - The owner is looking to automate this list! He said: “I want a list like the Netflix DVD queue that’s not really a queue.” (okay)

The less good news - Mr. Tee is old and quirky. The program 1) must run in the command line without graphics and 2) be written using the **C Programming Language**.

2. Requirements

Program #1 requirements are:

- ➢ Write your program in **C**.
- ➢ At the start of your program, read a **Videos file** (videos.txt) which describes the current video list. At the end of the program, save list changes to the Videos file.
- ➢ Implement the **video commands** listed below.
- ➢ Code your own **doubly-linked list** with a head and tail.
- ➢ Flex **your creativity**, add at least one own cool command/feature of your own.
- ➢ Follow our class **Coding Guidelines**, modified for C coding.

3. Details

Here are the video commands your program must support.

Command <args>	Description
`add <video>`	Add video to the list in a *random* position; the new video is new selected
`select <num>`	Select video at position <num>
`print`	Print video list
`up`	Move selected video up 1 position
`down`	Move selected video down 1 position
`first`	Move selected video to first position
`last`	Move selected video to last position
`reverse`	Reverse the order of videos in the list
`avalanche`	Move even position videos to the end
`remove`	Remove the selected video
`clear`	Clear all videos from the list
`exit`	Exit the program

The avalanche command is weird. Walk through the list and move each even-numbered video the the end of the list. Here's an example.

List: 1, 2, 3, 4, 5, 6, 7
List after avalanche: 1, 3, 5, 7, 2, 4, 6

The Videos file resides at videos.txt. The format is one video per line, in list order. Like this:

```
Manhattan
Casablanca
It's a Wonderful Life
Videodrome
Fargo
```

Simplify our UI and therefore string duties in C:

- Only 2 commands have parameters: add and select. Get the command, then ask for the parameter on a separate line. This way we won't have to parse strings.
- Your default action on errors should be to make something logical happen. Example: If user specifies "up" and the video is already first, just ignore it. If "remove" on an empty list, just keep going. In other words, go light on the error checking.
- Notice - commands are unique once you enter 2 letters. So, it's convenient for users (your testing!) to enter "pr" for print or "se" for select, and so on.

Little help?

- If you haven't coded in C before (cough), I have a helper file for you with goodies on the language and our assignment:

  C Programming Helper (gdoc)

- I'll post example sessions and test files elsewhere (k: drive or Piazza)
- We'll spend (a lot of) class time on this as well.

How to succeed (in any program):

1. Start early!
2. Don't be shy. Ask a question in class. Email me. Come to office hours.
3. Small bites. Divide your program into small, manageable tasks. Knock them down one by one.
4. Always be working. Your program should always compile and run. Never leave your work in disarray.

Grading

Create a **program1** folder on your k: drive.This folder should contain:

- All your C source files
- Your program1 executable
- Any test input and output files
- A **README.txt** file where you describe the status of your program and the creative command that you added

All your code must follow our class **Coding Guidelines**. Ugly code will be severely penalized. A program that doesn't even compile is probably worth 0 points.

Remember our **plagiarism** guidelines as well. Getting help from google or stackoverflow or a friend is OK, but:

1. You must acknowledge any help you receive with a comment in your code
2. You must understand any code in your solution
3. Get help on program components, not the primary assignment (the tic tac toe philosophy)
4. If you have any questions in this area, contact me **before** you turn in your work, not after (when it's too late)

thanks… yow, bill