CS117, Introduction to Programming III, C++
Spring 1998
ScriptEeze

Final Project, Due Friday June 12

The point of this final project is to put some of the search and retrieval algorithms discussed to fun and productive use. You'll write a program that takes in a movie or tv script and generates a hypertext document.

The program should first ask for the name of the script file then ask for the name of the script's *summary file*. Here the summary file will contain the title of the script and the characters. For example, the summary file for the "X-Files" script contains:

```
Title: Deep Throat
Characters:
FM
DS
DT
MrsB
ColB
MP1
MrsMcC
CK
PM
FW
```

Both script and summary files will be provided to you. After the program gets the file names, it will generate an html file which describes the script, and then will generate a hyperlinked version of the script in html.

Part 1: Script Analysis

Two files, `''hmtlUtils.h''` and `''htmlUtils.cp''` define some simple functions for generating html from C++. In this first exercise, you can either use your code from the last homework or use the STL *map* container to store the information that will be displayed in the html file.

The point of this part is to get familiar with the html generating functions. In the `''htmlUtils.cp''` file is a commented out *main()* function that generates a simple html file. You might want to play around with this to get an idea of how the utilities work. After you have familiarized yourself with the html generating functions, your first goal is to generate an html file that will display the following information about the script:

- **How many lines total are there in the script?** Here, let's let a "line" be one line of dialogue. So for example in the following *X-Files* dialogue:

```
MP1:     As soon as we're in position we'll be on a count of five.
(MPs walking through backyard, children's playground
  equipment)

MrsB:    (Pushing at people, breaking through police
barricade) Excuse me, thats my house.
```

  there are 2 lines. You can get a (approximate) count of the number of lines by counting the number of times that a line of the form

  `character-name:`

  appears in the text. You know who the characters are from the summary file. All you need to do here is count the number of times that the character has a line of dialogue.
- **A sorted list of the characters by the number of lines.** That is, a count next to each character showing the number of lines that they have in the script. You have the count from the the above information. Now you have to figure out how sort characters by the number of times that they speak.

You also should display qualitative information on the script based upon *word frequency.* For example, this means that in addition to keeping a record of the number of times that a character has dialogue, you should keep count of the occurrence of each word. Using this count, display in the generated html file:

- **Rating estimate?** Can you guess how the film would be rated given the frequency of partifular words that are found in the script? There's no right or wrong to this part. Think of a good heuristic judging rating based on the kind and type of words found in the text of the script. For example, you might populate a few `Set` containers with words associated that you think are more associated with R-rated or PG-rated films and then count the number of times those words are found in the script.

So in the end, the summary html file that you generate should look something like:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Movie Summary</title>
  </head>
  <body>
    <h1>Summary for Deep Throat</h1>
   Line Count: <br>
   Characters by Number of Lines: <br>
   FM: <br>
  </body>
</html>
```

This doesn't show the total line count and doesn't show the line counts for all of the characters. Each of the lines of html is readily generated by the utilities in ``htmlUtils.cp''.

Thought Question

Describe briefly (don't implement) how you could use a **trie** to do the rating estimate part of the last exercise. A brief description of tries is attached.

Part 2: Creating An Index

Once you have finished generating the summary file, now generate the hyper-linked document. First you should develop code that allows you to construct a simple index for the script. By index, I mean a list of words or phrases, followed by numbered occurrences in the script text. Each of the numbers should be *named anchors* that will allow you to go to the place where the word occurs in the text. The index should be put at the top of the hyper-linked script, and entries will link to words or phrases that occur later. Again, named anchors can be generated by the functions given in ''htmlUtils.cp''. So for example, the html from a segment of the *"X-Files"* index could look like:

```
<h1> Index </h1>
military <a href=''#military1''>1</a> <a href=''#military2''>2</a>
...
UFO <a href=''#ufo1''>1</a> <a href=''#ufo2''>2</a>
```

and later in the document

```
<a name=''military1''>Military</a> police dressed in SWAT gear.
...
 This isn't some <a href=''ufo2''>UFO</a> gooosechase...
```

When the above lines are viewed in a browser, the "1" and "2" appearing after "military" will be highlighted and click-able. You can go directly to the "Military police dressed in SWAT gear" line by clicking the "1" which follows "military" in the index. The "¡h1¿Index¡/h1¿" directs the browser to display Index as a section heading.

This isn't as awful as it seems. What you want to start off with is a set of words you want as indeci. I'll stipulate that there have to be at least 15 words in the index. You can either pre-designate them or cleverly determine them by your own analysis of the script.

You basically want your program to read a word from the old script and see if it's one of the index words. If it is you want to (1) generate a unique *fragment identifier* for the word (that's what the ufo2 and military1 are in the above examples), (2) store the fragment somewhere for when you generate the index, (3) then print the anchor text to the html file using the *writeHtmlNamedAnchor* function (this will print out the Military part) and (4) after the document has been done, print the index using the *writeHtmlAnchorRef* function (this will generate the military 1 part). One approach is to store the index in a map indexed by each work, with each map value as a list of fragments. When it comes time to print out the index, you'll have to (1) call the appropriate *ofstream* function to move to the top of the html file, and (2) iterate over the map by printing out first the word and then iterate over the fragment list, printing out each fragment using the *writeHtmlAnchorRef* function.

Part 3: Linking Lines

Now you want to have some way of jumping to the next line that a particular character speaks from the the line that you are at now. So for example, the following segment

```
DS:      You OK, Mulder?
FM:      Yeah.  I'm fine.

        *******************

        FBI headquarters.  Scully sitting at microfilm researching old
        newspaper stories.
        Heading "Sightings Reported"
        Title: "Ellens Air Force Base Mecca For UFO Buffs"
        Camera scans line of story "...technology with stealth weaponry..."

        ********************

        Mulder's apartment
        (phone rings)
FM:      Hello?
```

would be transformed to something like

```
<a name=''DS1''>DS:</a>  <a href=''#DS2''>Next</a>    You OK, Mulder?
<a name=''FM1''>FM:</a>  <a href=''#FM2''>Next</a>    Yeah.  I'm fine.

        *******************
```

This is basically a repeat of Part 2, only now you're adding the ability to navigate "inside" of the script instead of just from the index.