CSI 2131 File Management — Winter 2005
Computer Science — University of Ottawa

Homework Assignment #1 (100 points, weight 10%)

Due:

1 Program: Html-to-Text translator (90 marks)

1.1 Problem description

The objective of this exercise is to get you familiar with writing programs in C++, while applying some of the file processing operations seen in class.
You will write a C++ program that reads an input file in html (hyper-text markup language) format and writes an output file in text (ASCII) format. This will be similar to (but simpler than) what some web browsers do when you choose the option `Save as...` followed by option `text`. To see how an html file looks like, go to some web page and select a browser option of the type `View>Source`.
The html file contains many tags (special commands surrounded by `<>`), which will determine your output file formatting in the way specified below:

1. All of the following should count as a single space: several spaces in a row that separate words, any sequence of spaces and special characters for line change (CR+LF). Spaces not separating words such as the ones at the beginning of a line or between tags must be ignored.

2. There is no indentation at the beginning of a paragraph.

3. The line break tag `<br>` breaks the current line. Several of these in a row do accumulate.

4. The new-paragraph tag `<p>` breaks the current line (if not at the beginning of line) and skips another line. Several of these in a row do NOT accumulate.

5. Tags surrounding headings `<h1>Heading</h1>` (it can be `<h1>` up to `<h6>`):
`<hn>` produces the effect of `<p>` followed by $2n$ spaces for indentation; its closing tag `</hn>` produces the effect of `<p>`.

6. Ordered and unordered lists will produce indentation relative to the current indentation. Therefore, sublists will accumulate their indentation.
Beginning of list `<ol>` or `<ul>` will produce the effect of `<p>`.
List items `<li>` inside an unordered list `<ul>` produces the following 6 characters to be print after the current indentation: 4 spaces, 1 character `*`, 1 space.
List items `<li>` inside an ordered list `<ol>` produces the following 6 characters to be print after the current indentation: item number right justified in 4 spaces, 1 character `.`, 1 space.
A line break inside a list item will keep this 6-character indentation.

7. Link tags desired text produces: desired text <http:www.site.uottawa.ca>

8. Other tags should be ignored: skip them and they will have no effect in the text file.

Example input file:

```
<h1>Example of a piece of html file</h1> <p>The heading above
produced fine 2-space indentation. <br>  Spaces after
a line break have no effect.</p> <p>     Spaces are
also ignored at the beginning of paragraphs.</p>
<h4>Ordered and unordered lists</h4>
See how heading 4 produces 8-space indentation?<br>
Now I will show nested lists. <ul>
<li> Vegetables<ul><li>broccoli</li><li>aspargus</li>  </ul></li>
<li> Cheese <ol><li>Brie</li><li>Blue</li><li>
Gouda</li></ol> </li> <li> Cereal: corn, rice,<br>and barley</li>
</ul> Testing a <a href="www.site.uottawa.ca">link</a>.
```

Example output file

```
  Example of a piece of html file

The heading above produced fine 2-space indentation.
Spaces after a line break have no effect.

Spaces are also ignored at the beginning of paragraphs.


        Ordered and unordered lists

See how heading 4 produces 8-space indentation?
Now I will show nested lists.

    * Vegetables
          * broccoli
          * aspargus
    * Cheese
         1. Brie
         2. Blue
         3. Gouda
    * Cereal: corn, rice,
      and barley

Testing a link <www.site.uottawa.ca>.
```

1.2 Implementation details and standards for your program

Command line arguments:
The input file names for your program will be given on the command line. This means that after your program is compiled and an executable file is created, you are going to run it on the DOS prompt and specify some arguments which will represent file names to be used in your program. Details on how to access command line arguments inside your program will be explained in a separate page in the web. In short, the main mechanism to do that involves the use of the following arguments in the main program:
`int main (int argc, char* argv[])  ...`
You may look for information on the use of these parameters on the web or on a C++ book.

Creating and running your program:
The standards below (file names and way of running your program) should be followed strictly. You can use your creativity in the way you design your class and program, but the number and name of files and the way they are used by the teaching assistant should follow the standard.
Create a project and call it `html2text` with the following classes and files:

- HtmlTranslator class (files: `HtmlTranslator.cpp, HtmlTranslator.h`)
- main program (file: `main.cpp`)

The executable program will be called `html2text.exe` by default, since `html2text` is the project name.
This program (html2text.exe) will be executed at the DOS prompt where the user can specify command line arguments using the following format:
`html2text.exe <inputfilename> <outputfilename>`
or alternatively (the user may omit the extension .exe):
`html2text <inputfilename> <outputfilename>`
where `<inputfilename>` is the physical name of the input file and `<outputfilename>` is the physical name of the output file.

Testing and checking your output files:
We will provide input files for testing your program. We reserve the right of using additional input files when testing your program.When correct outputs are available for the given inputs, they will be provided to you at the webCT page.
Please, create the output file in the exact format that is specified in this handout, since it is likely that we will use some automatic way of comparing your outputs with correct ones.

Documentation and Style
Your program must be well-documented and written in good style. Part of the marks will be dedicated to documentation and style. To document your program, write comments explaining what is done in the following lines and add comments at the end of certain lines

explaining what has been done on that line. For each function or method add a full explanation before it, explaining its purpose and what its parameters represent. For each class created, add an explanation before it. Good style includes good documentation, choice of clear names (for variables, classes, functions, etc) and good program organization and design (like creating methods that deal with each aspect/task within a class).

What and how to submit your assignment
Please, read the course's policy on plagiarism and collaboration. You must not include ideas or pieces of code from your colleagues or from other programs available in the web or elsewhere. By submitting this assignment you are automatically acknowledging understanding of the policy.
Create a directory/folder named `a1` containing only the following files:
`main.cpp`, `HtmlTranslator.cpp`, `HtmlTranslator.h` and `written.txt`.
The file `written.txt` contains your answers to the written questions given in Section 2 and should be written in plain text (you may use notepad or similar editors). EVERY FILE MUST CONTAIN A HEADER WITH Student Name, Student Number, Course and Section. Zip the folder and submit the zipped file as your assignment#1 submission to webCT (only one file is submitted); this should be done in such a way that unzipping will produce folder `a1` with the files inside.

2 Written Part: Magnetic disks (10 marks)

Consider a magnetic disk with the following specifications:

- Capacity = 419,020,800 bytes
- Minimum (track-to-track) seek time = 1 msec
- Average seek time = 12 msec
- Spindle speed = 5200 rpm
- Average rotational delay = 6 msec
- Maximum transfer rate = 12 msec/track
- Number of bytes per sector = 512
- Number of sectors per track = 50
- Number of tracks per cylinder = 4
- Number of cylinders = 4092

and a file with 409,600,000 bytes organized in 1,600,000 records of 256 bytes each.
Please, show your steps in the next calculations. Calculate the time taken to read the whole file in two different ways, namely reading records **sequentially** or **in random order**, in each of the following cases:

1. The file is stored, in order, in contiguous cylinders and the next cylinder is filled after the previous one has been completely filled, track by track. Only in this case, consecutive sectors in contiguous cylinders are properly staggered so as to be read without rotational delay after the disk moves to the next cylinder.

2. The file is stored, in order, in cylinders randomly dispersed across the disk and the next cylinder is filled after the previous one has been completely filled, track by track.

3. The file is stored in tracks randomly dispersed across the disk and the next track is filled after the previous one has been completely filled.

4. The file is stored in sectors randomly dispersed across the disk.

5. **Challenging question:**
   Would the way your calculation was done, for randomly reading the file, change in any substantial way (due to rough approximations done before) if the file had only 800 records (same record size) in any of the above situations? If so, indicate in which of the previous 4 cases, explain in which way, and re-calculate with the the new data for the "most changed" case.