CS142/IT501 **Project 1** Fall 2015

Given: Mon Oct 12
Due: Fri Oct 30, 10 p.m.

This project is to transform the file viewer we created in class into a browser for a *web of files* that are all located on a single computer. (This is in contrast to the World Wide Web whose *pages* are located all over the world.)

Here are more details. The files are plain text files that may contain *anchors*. An anchor is a string of the form

<a *file text*>

where *file* is the name of another file and *text* is a piece of text. Note that an anchor must be separated from adjacent text by white space and that the strings *file* and *text* cannot contain white space. Here's an example of a line of text that contains a valid anchor:

```
Here is more <a info.txt information.>
```

Your browser should work pretty much like the file viewer we created in class, except that when an anchor is displayed, it should appear as

<<*text*>>[*number*]

where *number* is a number that is unique to this anchor. For example, the above line of text might be displayed as follows:

```
Here is more <<information.>>[17]
```

The browser should have an additional command *go* that asks the user for a number and then opens the file associated with the corresponding anchor:

```
After a few computer science <a courses.txt courses,>
students may start to get the feeling that programs can
always be written to solve any computational problem.
Writing the program may be hard work.  For example, it
may involve learning a difficult
<a difficult_techniques.txt technique.>  And many hours
of debugging.  But with enough time and effort, the
program can be written.

So it may come as a surprise that this is not the case:
there are computational problems for which no program
exists.  And these are not ill—defined problems (Can a
computer fall in love?) or uninteresting toy problems.
These are precisely defined problems with important
practical <a applications.txt applications.>
```

Figure 1: Sample file

```
command: g
link number: 17
```

In addition, your browser should format the contents of each file as follows. Each file is assumed to consist of paragraphs separated by blank lines. Each paragraph is a sequence of consecutive lines. Each line is a sequence of "words" separated by white space. Note that, in this context, a word may include punctuation. When a paragraph is displayed, all its words should be separated by a single blank space and arranged so that no line is longer than a length specified by the user. Paragraphs should be separated by one blank line. Figure 1 shows a sample file and Figure 2 shows how it should be displayed with the line length set to 40.

When launched, the program should ask the user for the maximum line

```
After a few computer science
<<courses,>>[1] students may start to
get the feeling that programs can always
be written to solve any computational
problem. Writing the program may be hard
work. For example, it may involve
learning a difficult <<technique.>>[2]
And many hours of debugging. But with
enough time and effort, the program can
be written.

So it may come as a surprise that this
is not the case: there are computational
problems for which no program exists.
And these are not ill—defined problems
(Can a computer fall in love?) or
uninteresting toy problems. These are
precisely defined problems with
important practical <<applications.>>[3]
```

Figure 2: Sample file of Figure 1 as displayed by the browser

length right after it asks for the window height.

The assignment policy available on the course web site also applies to this project. With one exception: if you do the project as a team, you are allowed to divide the work with your teammates. You will need to coordinate the work as a team, which essentially means that you should design the browser as a team, but a lot of the implementation work can be divided. For example, one teammate can write the formatting code while the other writes the code that deals with anchors and links.

If you work as a team of three, since this is not a very large program, it may be difficult to divide the work in three. In that case, two teammates could work together on one part of the program. And if you work as a team of two, you could also choose to do all the work together. But in these cases, make sure that every team member gets a chance to write some of the code.

Note that even if you divide the work with your teammates, you should expect to have to help each other out.

If you decide to work alone, I strongly suggest that you build the browser gradually. For example, you could start by dealing with anchors and links, and leave the formatting for later. Or the reverse. And even the formatting can be done gradually. For example, you could at first assume that the file contains a single paragraph. The important is not to try to do everything at once.

Hand in the following:

- *A design document*. This should be complete, precise and easy to understand. You can use the design document of the file viewer as a starting point. Make sure your program is highly modular.

- *All your code*. Make sure you write standard C++ code so I can compile it with Code::Blocks and g++.

- *A status report*. It should state whether your program works and, in case it doesn't work perfectly or is not complete, explain exactly what doesn't work or what remains to be done.