CSCI 1320 (Principles of Programming I), Spring 2012

Homework 7

Credit: 20 points.

1 Reading

Be sure you have read chapter 9.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course and the assignment (e.g., "csci 1320 homework 7" or "POP I hw7"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (10 points) Your mission for this problem is to write a Scala program that performs some of the same functions as the UNIX command `grep` — searches in a file or files for all lines that match a given search string, and prints the ones that do. For example, if you have a file containing the following lines:

   ```
   Hello world!
   There is a German word for worldview that I have forgotten.
   World without end.
   Goodbye!
   ```

   and you search for `world`, the first and second lines match. (It's easiest to do the match in a case-sensitive way, so the third line does not match.) `grep` by default prints each matching line preceded by the filename, so if the above text is in a file called `input1.txt`, and `input2.txt` is a file containing the lines

   ```
   world without end
   end of the world
   ```

   `grep world input1.txt input2.txt` produces the output:

   ```
   input1.txt:Hello world!
   input1.txt:There is a German word for worldview that I have forgotten.
   input2.txt:world without end
   input2.txt:end of the world
   ```

Like `grep`, your program should get the text to search for and the names of one or more files from its command-line arguments. So for example if you call your program `simple-grep.scala`, scala simple-grep.scala world input1.txt input2.txt should produce the same output as above. Your program should print an error message if there are not at least two command-line arguments. It's okay to assume that the files specified actually exist (and accept that if they don't, your program will crash); if that bugs you and/or you want an extra-credit point, you can write your program to instead print an error message about files that don't exist.

Hints:

- The "sum numbers from a file" sample program may be helpful as a way to get started.
- Scala `String`s have a method called `contains` that tells you whether one string contains another, e.g., if `s` is a `String`, `s.contains("hello")` evaluates to true if "hello" appears in `s` and false otherwise.

2. (10 points) Your mission for this problem is to write a simplified search-and-replace program in Scala, i.e., a program that, given an input file and two text strings (call them *oldtext* and *newtext*) and the name of an output file, searches through the input file and changes all occurrences of *oldtext* to *newtext* and saves the result in the output file. For example, given an input file with the following contents:

```
hello world from me
world hello hello world
just a line
hello!
```

changing "hello" to "HELLO" should produce a file that contains:

```
HELLO world from me
world HELLO HELLO world
just a line
HELLO!
```

Your program should get the names of the input and output files from command-line arguments and prompt the user to enter *oldtext* and *newtext*.

Hints:

- Scala `String`s have a method called `replaceAll` that that replaces all occurrences of one substring with another; e.g., if `s` is a `String`, `s.replaceAll("foo", "bar")` produces a new `String` identical to `s` except that "foo" has been replaced with "bar".

For extra credit (up to 5 points), have your program do an interactive search and replace. You have two options (the second of which is more difficult and therefore worth more points):

- Print each line that contains *oldtext* and ask the user whether to change *oldtext* to *newtext* everywhere in that line.
- Ask separately for each occurrence of *oldtext* (e.g., in the sample input above, the program would prompt twice for the line that contains "hello" twice). If you choose this option you may find two additional `String` methods helpful:

- `indexOf` takes a string to search for and a starting index and returns the starting position of the text to search for, or -1 if it is not found at or after the starting index. (E.g., `"hello".indexOf("ll", 0)` returns 2, while `"hello".indexOf("ll", 4)` returns -1.)
- `substring` takes a starting and ending index, or just a starting index, and returns a substring of the original string. (E.g., `"hello".substring(0, 2)` returns `"he"`, while `"hello".substring(2)` returns `"llo"`.)