COMPSCI 105: Principles of Computer Science
Summer 2007

Tutorial Three: I/O

Helen Gu and Slobodan Vukanovic
{ygu029, svuk002}@ec.auckland.ac.nz

This tutorial is not being assessed. It provides you with an opportunity to become familiar with concepts introduced in lectures.

Question One: Copying Files

Write a `fileCopy ()` method that takes in names of input and output files as arguments and copies the contents of the input file to the output file.

```
void fileCopy (String input, String output) {

   /* Answer given using file readers and writers. Can also use
    * buffered readers and writers, or print writers.
    */

   try{

      FileReader in = new FileReader (input);
      FileWriter out = new FileWriter (output);

      int currChar;
      while ((currChar = in.read ()) != 1)
         out.write (c);

      in.close ();
      out.close ();

   }

   catch (FileNotFoundException e) {
      e.printStackTrace ();
   }

   catch (IOException e) {
      e.printStackTrace ();
   }









}
```

Question Two: Directory Information

We would like to know the following information about files in a directory: name of each file, its type (i.e. whether it is a file or a directory), its size in bytes, and the date the file was last modified. This information is to be kept inside the directory whose files we are interested in, in a file called `dinf` (stands for ``directory information'').

Download `DirInfo.java` from the course website (look under ``Tutorials''). You need to complete the `processDir ()` method. The method creates the `dinf` file in the right place, steps through the directory given as an argument when you run the program, and updates the `dinf` file. For example, the test directory `TestDir` has the following files in it: `AnotherTestDir`, `HelloWorld.c`, `HelloWorld.java`, `HelloWorld.py`, and `HelloWorld.rb`. The output of running `DirInfo.java TestDir` from the command line creates a `dinf` file inside `TestDir`, contents of which should look something like this:

```
Name                Type    Size       Last Modified

AnotherTestDir      dir     136        2007-01-14
HelloWorld.c        file    66         2007-01-14
HelloWorld.java     file    114        2007-01-14
HelloWorld.py       file    22         2007-01-14
HelloWorld.rb       file    21         2007-01-14
```

Don't worry too much about formatting the output. The important thing is that it contains the relevant information. **Note that `dinf` does not contain information about itself**. You can download `TestDir.zip` from the same place you found `DirInfo.java` to test your code on.

Here's further information about `DirInfo.java`, in case you can't get started:

The `main ()` method processes the command line arguments and either starts the program, or prints usage information (in case something went wrong).

The `path` variable stores the path to the directory. It is updated from the command line.

The `FILENAME` constant stores the name of the `dinf` file, which is just ``dinf''.

The constructor invokes `processDir ()`, which you are required to complete.

Pseudocode of the `processDir ()` method is:

```
processDir ():


   create the dinf file

   print header of dinf file

   for each entry in directory pointed to by path:
      get name, type, size, and date last modified of entry
      write the above information to dinf

   close dinf
```

Question Three: Word Replacement

Complete the word replacement program `Replace.java` from the course website (look under ``Tutorials''). The program should take in an input file, an optional output file, and some words from the command line. Words are given as a list of pairs, where each pair contains a word to be replaced and a replacement word. For example, if you had the following text in a file called `text` (note that punctuation marks are assummed away):

```
assume that ernie and bert want to send coded messages ernie gives bert
his public key assuming that the key was not intercepted and replaced
with someone elses key bert can now send data to ernie securely because
data encrypted with the public key can only be decrypted with the
private key which only ernie has

Text taken from Kernel Programming Guide Apple Computers Inc
```

and ran `java Replace text -o replaced Apple Orange bert snert ernie hagar` from the command line, the program should replace ``Apple'' with ``Orange'', ``bert'' with ``snert'', and ``ernie'' with ``hagar'', and print the output to the file called `replaced`:

```
assume that hagar and snert want to send coded messages hagar gives
snert his public key assuming that the key was not intercepted and
replaced with someone elses key snert can now send data to hagar
securely because data encrypted with the public key can only be
decrypted with the private key which only hagar has

Text taken from Kernel Programming Guide Orange Computers Inc
```

When the -o option is omitted, the input file also serves as the output file, i.e. after the program runs, the contents of the input file are overwritten with the program's output.

You need to complete two methods. `mainLoop ()` is the main loop of the program, which goes through the input file line by line and updates the output file. `replace ()` takes a string representing a line in the file, and returns a string representing the modified line. Here is the pseudocode:

```
mainLoop ():

   Create reader and writer for input and output files respectively

   for each line in the input file:
      replace () the current line
      write the replacement line to the output file

replace (line):

   for each word in line:
      getNewWord () for the word
      append the new word to the new line
   return new line
```

The only tricky bit is when the input file and the output file are the same. We cannot open a file for reading and writing at the same time. This is solved by creating a temporary file whose name is the same as the name of the input file plus a `temp` suffix. When the main loop finishes, you need to remove the input file and rename the temporary file with the name of the input file.

The rest of the program looks like this:

`fileIn` is the name of the input file.

`fileOut` is the name of the output file.

`overwrite` is a flag which is set when -o option is not given, i.e. when the input and output files are the same.

`oldWords` is an array of old words, i.e. words to be replaced.

`newWords` is an array of new words, i.e. replacement words for `oldWords`. `newWords[i]` is a replacement for `oldWords[i]`.

`getNewWord ()` returns a replacement word for a word passed in as argument. If the word is not to be replaced, the method returns the argument.

`main ()` method starts the program.

`checkArgs ()` is a method that parses the command line arguments and sets the flags.

`printUsage ()` prints usage of the program. Invoked when things go wrong.