Cs352 — Homework #6
C-shell scripts and awk (Version 3)

November 5, 2003

Due Time: 11/6/03 (9:00PM). Submission in pairs is **NOT** allowed.
Turnin ID: cs352_assg6
Turnin Files: indent.awk lines_summation check_hw

1. Create an awk file, called `indent.awk` in order to indent properly a C source file. You can assume that some command starts, or terminates with curly brackets ('{' and '}'). In the output, the text after hitting a '{' sign should be indented by 3 blanks to the right, with respect to the previous line. Similarly indent left after hitting a '}' sign.

   To be more detailed. A '{' in the current input line might appear anywhere in the current line of the input file, and effects only lines after the current line. A '}' appears only at the beginning of the line, and effects also the current line and the lines after the current line.

   Example:

```
if (x==y) { /*This line is not indented */
        printf(``Hello world\n'');
        } /*This line should be idented left */
```

   You can assume that comment lines do not contain '{' or '}', and that a line contains at **most one** '{' or '}' sign. Also between any two consecutive words of the input (separated by one or more blanks in the input) there must be exactly one blank between the two words in the output.

For example, if the input is

```
#include                    <stdio.h>

        main(   )  {
   int i       ,  j  ;
  for(i =   1  ; i <  10  ; i++   ) {
     for(  j =   1  ; j <  10  ; j++   ) {
     printf  ("%d", i *  j );
     }
     }
     }
```

Then the output should be

```
#include <stdio.h>

main( ) {
   int i , j ;
   for(i = 1 ; i < 10 ; i++ ) {
      for( j = 1 ; j <  10 ; j++ ) {
         printf ("%d", i *  j );
      }
   }
}
```

2. Write a C-shell script called `lines_summation` that prints the total number of lines (as reported by the utility "wc") of all files which are executables, in the directory from which the script is invoked. Don't do this recursively to the files in the subdirectory. The only output of the program is the number of lines. So if in the current directory there are 2 executable files, containing 12 and 20 lines (respectively), then the output of the command `lines_summation` is 32. If there are no executables in the current working directory, simpley print 0 as the result.

3. Write a C-shell script file, called `check_hw`, which accepts a list of parameters. For example,
`check_hw executableFile inp1 out1 inp2 out2 inp3 out3 ...  .`

The first parameter (executableFile in the above syntax representation) is a name of program to be checked. If there is no file whose name is the same as the first parameter, an error message "No executable found" should be printed and the whole script terminates. Next the script checks whether there exists a **readable** file inp1 (check that it exists and is not a directory, otherwise we regard this testcase as a failure and its index number will be in the output). If yes, the scripts calls executableFile, and redirect its input to be taken from the file inp1. The script checks if the output in this case is identical to the contents in file out1 (check that it exits and is not a directory, otherwise we regard this testcase as a failure and its index number will be in the output). If this is not the case, we say that executableFile **failed** on testcase inp1 and its index number will be in the output. Next, the script runs executableFile on the input files inp2, inp3 etc, and checks if the output is identical to out2, out3 etc. (note that the number of input files is not limited, but the input files and output filenames specified on the "check_hw" command line are always in pairs, though some of the files specified may not exist on disk).

[**Bonus (**%5**):** `However, because executing executableFile on certains inputs might caused infinite loop, you must ensure that the running of exeutableFile would be foreced to terminate when it runs for roughly 5 seconds CPU time and is still running. We say that in this case (the executable is forced to terminate) executabeFile` **failed** `when running on this input which causes the infinite loop.]`

The output of the script is a list of the index numbers of the testcases that executabeFile failed on. So for example, if we call the script as follows:

```
check_hw hw17 file1.inp file1.out file2.inp file2.out file3.inp file3.out
file4.inp file4.out
```

and hw17 once reading from file1.inp created an output identical to the content of file1.out, while file2.inp caused infinite loop, and file3.inp created output different from file3.out, file4.out is a subdirectory, then the output of the script is:

Output:

2 3 4

Another example is that if there is no executable in the current working directory that has the same name as the first argument passed to *check_hw* (*myprg* in the following example), you should print out the following error message and terminate the script *check_hw*.

```
check_hw myprg file1.inp file1.out file2.inp file2.out
```

Output:

No executable found

When no input or output filename is specified on the command line, you still need to check if the executable exists. If it exists, print nothing since there is no testcase to be tested (input or output files) at all. You can assume, the executable filename will always be specified on the command line. In the following example, executable "myprg" exists. So the output is just nothing.

`check_hw myprg`

Output: