# Cs352 — Homework #9
# C-shell scripts, awk and a bit signales

April 25, 2004

Due Time: 29/4/04 (9:00PM). Submission in pairs is **NOT** allowed.
Turnin ID: cs352_assg9
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. 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. Moreover, if a at you input, you reach a '}' with does not have a matching '}' proceeding it, an error message */* *THIS* } *IS IGNORED* */* should be printed after the '}', and it should not affect the indentation.

   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 );
      }
   }
}
} /* THIS } IS IGNORED */
```

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.

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 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 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 files are always

in pairs). 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, or 10 seconds clock time (consider using the command 'limit') 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

4. Change the program 'SignalDemo' studied in class, so that pressing ĉ twice within 0.5 second would caused it to exit. The source code can be found in http://www.cs.arizona.edu/classes/cs352/spring04/demos/

5. Write a script file named `MakeHist` that reads a file `numbers.inp` where each line contains a single integer in the range 1 to 99. The script should print a histogram of these numbers, using steps of 10. So the first line of the output is the string "0 :", followed by a sequences of $n_1$ star signs '*', where $n_1$ is the number of times that numbers in the range 0 to 9 appears in `numbers.inp`. The next output line is is the string "10:", followed by a sequences of $n_2$ star signs '*', where $n_2$ is the number of times that numbers in the range 10 to 19 appears in the input file. and so on. For example If `numbers.inp` contains the numbers
   99
   66
   88
   61
   7
   99
   99
   99
   Then the output is:
   0 :*
   10:
   20:
   30:
   40:
   50:
   60: **
   70:
   80: *
   90: ****

   Instruction. Your script file should first sort the input, and then called the awk file `MakeHist.awk` on the sorted file.