Exercise Sheet #3

Bulcsú Sándor <sandor@th.physik.uni-frankfurt.de>
Hendrik Wernecke <wernecke@th.physik.uni-frankfurt.de>
Laura Martin <lmartin@th.physik.uni-frankfurt.de>
Christopher Czaban <czaban@th.physik.uni-frankfurt.de>

Reminder: How to submit your solutions

(a) programms must be fully commented and compilable

(b) file names must contain your name and the problem number

(c) subject of submission email must contain `CPP` and problem sheet

Problem 1 (*Some file processing in Linux*) 6 Pts

(a) Renaming a large number of files (3 Pts)
Sooner or later you might find yourself in a situation where you have to handle or manipulate large amounts of files. A possible scenario could be the renaming of many files. Doing this file by file would result in a big waste of your valuable working time. Fortunately the Linux shell is a very strong ally for (not only) such kind of tasks.
`Your task:` Write a script that contains a loop (e.g. for or while) which produces 10 files with the following name pattern:

```
data_file_0000.dat
...
data_file_0010.dat
```

Having done this write another loop which preserves a part of the file name but removes the file_ part, i.e.

```
data_0000.dat
...
data_0010.dat
```

Hint: One possible way to replace a substring of a string stored inside a variable is:

```
MY_VAR="nice_weather"
MY_VAR=${MY_VAR/nice/bad}
echo $MY_VAR
```

(b) File processing with awk (3 Pts)

Awk is an own language shipped with most UNIX systems and is extremely useful to produce reports from or filter large amounts of raw data of any kind. Routines in awk are simpler and quicker to compose and much shorter than it is the case for other conventional languages. In certain aspects it is very similar to the C-language and it understands the same arithmetic operators, hence it can be considered to be a pseudo-C interpreter. The following exercise has the purpose to cover some very basic functionalities of awk.

`Your task:` Create a file with the following lines of numbers:

```
//file matrix.dat
1       0.5   0.25   0.125   0.0625
10      1     5      3.8
100     1.5   1
1000    2     4.9    1       10.9
10000   2.5   999    23      1
```

Write an awk script that ...

- computes the mean value for every row,
- the mean value of the first column,
- prints out rows with a mean value ≥ 1 and ≤ 50,
- computes the trace (imagine the data to be a quadratic matrix with some missing entries).

It is up to you if you combine all of these functionalities in a single script or rather write a single script for every functionality.

A brief introduction and some hints:

Awk processes files line by line and a script has the following general structure, consisting of three parts:

```
BEGIN {  }  #Action taking place before the
            #first line of the file
      {  }  #Action taking place for each line
END   {  }  #Action after the last line
```

Every part by itself is optional which means you could remove it entirely if not needed. In order to grasp this concept write a script with the following awk program, execute it on some arbitrary file and see what happens:

```
awk 'BEGIN {print "START"  }
           {print  }
     END   {print "STOP" }' filename
```

You can even use awk without processing a file. Some examples are:

```
awk 'BEGIN {print "Playing with numbers: ";
            print "10/5*3+3 = " 10/5*3+3;}'
awk 'BEGIN {for(i=1;i<=10;i++){print "i = "i}}'
awk 'BEGIN {my_variable="10"; print my_variable/2;}'
```

As you can see code has to be enclosed by curly brackets. Unquoted strings are interpreted as variables. Accordingly when you intend to print a string literally you have to put it in quotes. Awk's main purpose is processing files and operating on its lines and fields. Awk's standard field separator is white space, i.e. if you have a file like the one above every number is treated as a single field. For instance if you want to print the first and the second field or column, respectively, of every line you would simply use the following awk code:

```
awk '{print $1 " " $2}' filename
```

You can also format your output with the `printf` function which works exactly as its equivalent from the C-language. Information about the line number and the number of fields per each line can be accessed through the variables NR (number of records) and NF (number of fields), as with the following program:

```
awk '{print "Line nr. " NR " has " NF " columns ";
     }' filename
```

Further control flow operations are pretty equivalent to the C-language as well, for instance if-cases:

```
awk '{if($1>10) print $1;
     }' filename
```

To give you a rather complete example of an awk script consider the following file, listing the employees of a smaller company including their position, monthly incomes and working hours per day.

```
//file person_income_working_hours.dat
Newman   manager             10000   16
Johnson  senior-consultant   8000    14
Loyd     consultant          6500    14
Walker   consultant          6000    14
Jackson  consultant          6200    14
Bering   intern              1200    20
```

The following awk script computes the average income of a consultant working for this company. Furthermore it calculates the average and total working hours of all employees.

```
awk 'BEGIN{
```

```
            nr_consultants=0;
            average_income_consultant=0;
            working_hours=0
          }
          {
            if($2=="consultant")
            {
                nr_consultants++;
                average_income_consultant+=$3
                print $2
            }
            working_hours+=$4;
          }
    END{
            average_income_consultant/=nr_consultants;
            print "average_income_consultant="
               average_income_consultant;
            print "total working hours: " working_hours;
            print "average working hours: " working_hours/
               NR;
       }' person_income_working_hours.dat
```

By now you should have sufficient information to accomplish the exercise. If not feel free to search the internet for further information, e.g.:

`http://www.grymoire.com/Unix/Awk.html`

Problem 2 (*Code snippets*) 8 Pts

Embed the following code snippets into a running program, complete the code where neccessary. Mind that the snippets might also contain bugs (i. e. mistakes). (In general it can be beneficial to use additional compiler flags like `-Wfatal-errors -Wconversion -Wall -pedantic`. This can protect you from using bad programming practices with which your program might still compile but could provoke subtle errors at some point that can not be caught by the compiler. Use the man pages (`man g++`) to look up their meanings and test them.)

(a) (2 Pts)

```
int myArray[5] = {1, 3, 5, 7, 9};
for (i=0 ; i<=6 ; ++i) {
  cout<< "myArray[" << i << "] = " << myArray[i] << endl;
}
```

(b) (2 Pts)

```
double f, x
x = 1;
f = sin(x);
printf("sin( %.4f ) = %.4f", x, f);
```

(c) (2 Pts)

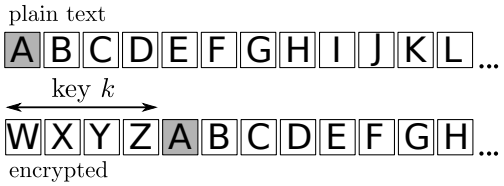
```
void divWithRem(int a, int b) {
  int h = a/b;    / auxiliar variable
  cout << a < " / " << b << " = " << h;
  cout << ", remainder: " << a%b << endl;
}
divWithRem(7, 2.);
```

(d) (2 Pts)

```
double myfunc(double a, double b) { return a*a+b*b; }
int result = myfunc(5, 2);
```

Problem 3 (*Caesar cipher*) 6 Pts

The Caesar cipher is a so-called single-alphabet substitution cipher and one of the most simple encryption algorithms. In order to encrypt a certain letter by this cypher you choose a key $k \in \{0, 25\}$ and shift the letter by k letters in the alphabet. E. g. if you choose $k = 4$ then E is encrypted by A.



Implement a program which asks for a sentence, if you want to encrypt or decrypt and which key you want to use. With the function `getline()` the program is able to read out more than one word. To write a coding algorithm it is useful to know more about elements of type char. Each character is assigned to an index. By increasing or decreasing the index it moves up or down in the ascii-alphabet. E.g. `char letter = 'A'` is assigned to 65. By increasing `letter+=1` the letter `'B'` is stored in the variable. You should transform all letters into capital letters (`toupper(int)`) for simplification. The strings can contain spaces and special characters which should not be encoded. The function `isalpha(int)` returns true only if the character is an alphabetic letter.