Assignment 2 - Software I, Spring 2004 (0368-2157-09/12)

`http://www.cs.tau.ac.il/~efif/courses/Software1_Spring_04`

Due: April 20, 2004

Required Knowledge rand, srand, arrays, functions, recursion
Before starting to answer the questions, please read very carefully the "Submission Guidelines"[1].

Ex 2.1 dice.c

Modify the dice-rolling program as follows:

1. simulate the rolling of 3 dice instead of 2.
2. compute the floating-point average μ of all the rolls of the 3 dice

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

3. compute the standard deviation s, which is expressed as

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

You may use the sqrt() function from the math library.

The program should read the random-generator seed and the number of tries, and print out the average and the standard deviation, followed by a histogram. Use `double` type to represent any real number in the program, and print out the result with 6 digits of precision for the fraction part. Each line of the histogram has the following format: sum of rolls, followed by a colon, followed by a space, followed by the count in parenthesis, followed by stars depicting the count. The count is aligned to the right, and truncated to the 5 least significant decimal-digits, or padded with spaced. The number of starts in the longest sequence should be 20, and the number of starts in the other should be normalized accordingly, and rounded to the nearest integer, or down in case of a tie.

[1] `http://www.cs.tau.ac.il/~efif/courses/Software1_Spring_04`

```
dice
10
10000
10.485600
2.978388
 3: (   46)*
 4: (  141)**
 5: (  291)*****
 6: (  455)*******
 7: (  713)************
 8: ( 1010)*****************
 9: ( 1166)*******************
10: ( 1223)********************
11: ( 1209)********************
12: ( 1105)******************
13: (  985)****************
14: (  721)************
15: (  469)********
16: (  292)*****
17: (  123)**
18: (   51)*
```

Ex 2.2 det

Write a program that reads a positive integer number n, followed by an $n \times n$ matrix, and computes the determinant of the matrix.

You may assume that $n < 10$. The matrix is represented as n rows of n integers in each row, the integers are separated by one or more spaces, each row is terminated by a newline. (Do not use a multi-dimensional array to store the matrix.)

Use a recursive method for computing the determinant. Suppose you wish to find the determinant of a 4×4 matrix. Then each element in the first row of this matrix is to be multiplied with a suitable 3×3 determinant. But while evaluating the determinant of a 3×3 matrix, we again need to multiply the elements of the first row of the 3×3 matrix by the appropriate determinants of the 2×2 matrices. That is where recursion comes in. The sign (+/-) should change for every alternate element. The logic used here is: to evaluate an $n \times n$ determinant, we multiply the first row elements of this determinant with a corresponding $n-1 \times n-1$ determinant by first copying the $n-1 \times n-1$ determinant into a separate array. We continue this process recursively until we reach a 1×1 determinant and then retrace.

For example:

$$\det \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = 1 \times \det \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} - 2 \times \det \begin{bmatrix} 4 & 6 \\ 7 & 9 \end{bmatrix} + 3 \times \det \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} = 0$$

Ex 2.3 palindrome

Write a program that reads an input line, determines whether it is a palindrome or not, and prints out "true" or "false" accordingly.

A palindrome is a string that is the same when read backward or forward.

Disregard spaces, non alphabetic and numeric characters, and ignore case distinctions. For example, the following are palindromes:

```
Was it a rat I saw?
Dennis and Edna sinned.
step on no pets
A Fool, A Tool, A Pool; LOOPALOOTALOOFA!
Madam, I'm Adam.\\
A man, a plan, a cat, a ham, a yak, a yam, a hat, a canal-Panama!
```

The program should disregard characters beyond the first 80, and terminate upon EOF as well as the end of the line.

Ex 2.4 rec_string

A *null-terminated* array of type char is called a *string*. Null-terminated means that the array contains a special character ' \0 '. The sequence of characters from the beginning of the array up to, but not including, the special character ' \0 ' is the content of the string. The standard C library `stdlib.h` manipulates strings. It contains the following functions:

1. Return string length
   `int strlen ( const char * string );`
   Returns the number of characters in `string` before the terminating null-character.

   string - null-terminated string.
   return value - the length of `string`.

2. Copy string
   `char * strcpy ( char * dest, const char * src );`
   Copies the content pointed by `src` to `dest` stopping after the terminating null-character is copied. The array `dest` is assumed (without checking) to be long enough to contain the `src` string.

   dest - destination string, assumed to be long enough to contain `src`.
   src - null-terminated string to copy.
   return value - `dest` is returned.

3. Append string
   `char * strcat ( char * dest, const char * src );`
   Appends `src` string to `dest` string. The terminating null character in `dest` is overwritten by the first character of `src`. The resulting string includes a null-character at the end. The array `dest` is assumed to be long enough to contain `src` string.

   dest - a null-terminated string long enough to contain both `src` and `dest`.
   src - null-terminated string to append.
   return value - `dest` is returned.

4. Compare two strings
   `int strcmp ( const char * string1, const char * string2 );`
   Compares `string1` to `string2` character by character. This function starts comparing the first character of each string. If they are equal to each other continues with the following pair until the characters differ or until end of string (the special character ' \0 ') is reached.

   string1 - null-terminated string.
   string2 - null-terminated string.
   return value - returns a value indicating the lexicographical relation between the strings:
   < 0 - `string1` is less than `string2`
   0 - `string1` is the same as `string2`
   > 0 - `string1` is greater than `string2`

Implement these functions using recursion, without using loops. The use of any functions from the `string` library is forbidden (do not include `<string.h>` in your source file, to avoid confusion). Place the code of all the four functions in a single file `rec_string.c`.

To test your code, follow the steps below. This is optional, and provided only for your convenience. The concept of makefiles will be explained later in the course.

1. make sure that `rec_string.c` does not contain the `main` function.
2. make sure that the file `rec_string.c` compiles successfully.
3. grab the files below from the assignment web-page and place them in the same directory where `rec_string.c` resides: `test.c`, `makefile`, and `test.in`.
4. type:

   ```
   make
   ```

   This command will compile the source files `rec_string.c` and `test.c`, link their object files and produce an executable file called `test`, run the executable with the input `test.in`, and compare the output using `diff` with the output obtained when the standard string-functions are used instead.
5. you can modify `test.c` and `test.in` to enhance the testing
6. to remove the intermediate files, type:

   ```
   make clean
   ```

Very important: your submission must include the single file named `rec_string.c` containing all four functions. It **must not** contain a `main` function!

Good Luck!

More Information on the Submission

Files Name

The files for the exercises should be located under `~/software1/assign2`, and their names should match the name of the exercise. Note that names are case sensitive (i.e. `ex1.C` is different than `ex1.c`).

Giving Permission to the Files

Before submitting the solution set, please give permission to the files by executing the following command:

```
chmod 705 ~ ~/software1 ~/software1/assign2 ~/software1/assign2/*
```

Sample Files

In order to test your programs, we have prepared some input and output files which you may compare to your own output. You may use the command `diff` or `diff -b` in order to compare your files to the sample files.

In order to read the input from a file and to write your output into a file you should use redirection. For example:

```
det < infile > outfile
```

The input and output data files may be found in the assignment web-page. The names of these files match the names of the corresponding programs, where the input files have a `.in` extension, and the output files have a `.out` extension.