Einführung in die Programmierung für Physiker

SoSe 2024 – Marc Wagner

Michael Eichberg: eichberg@itp.uni-frankfurt.de

Exercise sheet 8

To be handed in on 14.06.2024.

Exercise 1 [*Parsing a text file containing decimal numbers*] (4 + 4 = 8 pts.)

In Germany, we typically use a comma as a decimal separator, instead of a point. When writing a C program for reading a text file containing decimal numbers, where a decimal comma is used, we have to replace it with a point, to be able to convert it with `atof`. Moreover, when extracting a list of numbers from a text file, there might be several valid choices for a separator. Your task is now to write a program, which can parse such a list and store the numbers in a `double` array.

(i) Write a `main` function which takes command line arguments. The arguments are optional and thus need not necessarily be provided. Define variables corresponding to the options

- decimal separator (`char`), default is `"."`,
- list element separator (`char`), default is `" "`.

Then, define a function which parses the command line arguments and changes the values of the above variables accordingly.

- If argument number i is `"-d"`, argument number $i+1$ provides the decimal separator,
- if argument number i is `"-v"`, argument number $i+1$ provides the list element separator.

The decimal and list element separators are not allowed to be equal. Furthermore, the list element separator is not allowed to be a point. Print an error message and terminate the program, in either case.

(ii) Define a function, which reads in a file from `stdin` and in the first step replaces all decimal separators with points. In the second step, replace all list element separators with `" "`. Using techniques discussed in the lecture, convert each decimal number to `double`, store them in a dynamically allocated array and print them to the terminal. Remember to free the memory at the end of your program.

Time to Test! Download the following lists:

`https://itp.uni-frankfurt.de/~eichberg/pprog-sose2024/numbers1.txt`,
`https://itp.uni-frankfurt.de/~eichberg/pprog-sose2024/numbers2.txt`.

Run the program on the first list using the command line arguments `-v "-"` (not using the `-d` option) and on the second list using `-d "," -v "/"`.

Exercise 2 [*Matrix determinant*] (2 + 3 + 3 + 3 + 1 = 12 pts.)

In this exercise, we want to use Laplace's determinant expansion to compute determinants of matrices of arbitrary size. We will also split our program into several functions, each taking care of independent tasks, to keep our code readable and flexible.

(i) Multi-dimensional arrays, e.g. two-dimensional arrays representing matrices, are not always practical in C. As an alternative, one can define a superindex, which maps several indices to a single index. For an $N \times N$ matrix, the superindex can be defined as

$$a = i \cdot N + j \,,$$

where i is the row and j the column index. From a and N, one can then determine i and j using the modulo operator. Write a function which dynamically allocates memory for an $N \times N$ matrix, i.e. N^2 times the size of `double` bytes. Also write a corresponding function, which frees the memory. Then, write two functions realizing the concept of a superindex, which map $(i, j) \mapsto a$ and $a \mapsto (i, j)$.

(ii) Write a function which generates the matrix minor $\bar{A}_{ij} \in \mathbb{R}^{(N-1)\times(N-1)}$ of $A \in \mathbb{R}^{N\times N}$, defined as

$$(\bar{A}_{ij})_{kl} = \begin{cases} A_{kl} & \text{if} \quad k < i, l < j\,, \\ A_{k+1,l} & \text{if} \quad k \geq i, l < j\,, \\ A_{k,l+1} & \text{if} \quad k < i, l \geq j\,, \\ A_{k+1,l+1} & \text{if} \quad k \geq i, l \geq j\,. \end{cases}$$

Use your functions for memory allocation and superindices from task (i).

(iii) Laplace's expansion to compute the determinant of a matrix $A \in \mathbb{R}^{N\times N}$ with respect to the ith row reads

$$\det A = \sum_{i=0}^{N-1} (-1)^{i+j} \cdot A_{ij} \cdot \det \bar{A}_{ij}\,.$$

Implement this formula to compute $\det A$ for $j = 0$ in a recursive way in your program. Free memory allocated for matrix minors as soon as they are not needed anymore.

Time to Test! Compute the determinant of a 3×3 rotation matrix

$$A_1 = \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix},$$

where the Euler α, β, γ angles are initialized with several values of your choice (keep in mind that angles have to be provided in radians; you can use `M_PI` from `math.h` to convert degrees to radians), as well as the matrix

$$A_2 = \mathrm{diag}(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)\,.$$

(iv) Split your code into two source (`.c`) and one header (`.h`) file. One file should contain only the `main` function (named e.g. `exercise_XX.c`). The remaining functions are declared in `determinant.h` and defined in `determinant.c`. Where do you include what? Then, first compile `determinant.c` into an object (`.o`) file. After that, compile your full `main` function and link it to the previously generated object file.

(v) Add a `printf` statement to your allocation and deallocation functions. Without recompiling `deteminant.c`, but using `determinant.o`, compile your `main` function again. Do the `printf` statements take effect when you execute the program? Recompile `determinant.c` into an object file and try again. How about now?