Dirk Praetorius,
Marcus Page

Wintersemester 2011
11. Nov. 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 5

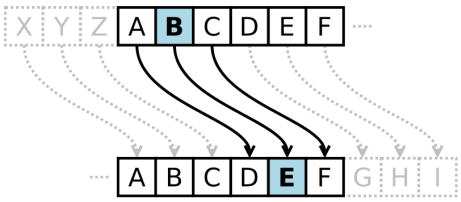
Aufgabe 5.1. Schreiben Sie einen Struktur-Datentyp `Umatrix`, zur Speicherung von oberen Dreiecksmatrizen $A \in \mathbb{R}^{n\times n}$. Dabei bezeichnet man eine Matrix

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & \ddots & & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix}$$

als obere Dreiecksmatrix. Mathematisch formuliert gilt also $U_{jk} = 0$ für $j > k$. Die offensichtlichen Nulleinträge sollen hierbei nicht gespeichert werden. Sie müssen sich also vorab überlegen, wieviele Einteäge Sie in den jeweiligen Zeilen haben. Schreiben Sie außerden Funktionen `newUmatrix` und `delUmatrix`, welche eine obere Dreickecksmatrix allokieren und initialisieren, bzw. den Speicher wieder freigeben. Schreiben Sie zusätzlich Funktionen `plotUmatrix` und `readUmatrix`. Diese sollen eine Matrix von Typ `Umatrix` auf dem Bildschirm ausgeben, bzw. über die Tastatur einlesen. Speichern Sie den Source-Code unter `umatrix.c` in das Verzeichnis `serie05`.

Aufgabe 5.2*. Schreiben Sie eine Funktion `solveUmatrix`, die ein Gleichungssytem der Form $Ux = b$, wobei U eine obere Dreiecksmatrix ist, löst und den Ergebnisvektor x zurückgibt. Ein aufrufendes Hauptprogramm soll U und b von der Tastatur einlesen und die Lösung x ausgeben. Verwenden Sie hierzu `readUmatrix`. Schreiben Sie außerden Funktionen `getEntry` und `setEntry` mit denen Sie konkrete Werte der Matrix verändern können. Speichern Sie den Source-Code unter `solveUmatrix.c` in das Verzeichnis `serie05`.

Aufgabe 5.3*. Die Cäsar-Chiffre ist ein Primitiver Algorithmus zur Verschlüsselung von Texten. Als Schlüssel wählt man hierbei eine Zahl $z \in [1, \dots, 25]$ die angibt, um wieviel der Klartext für die Chiffrierung 'verschoben' wird. Ausgehend vom Klartext verschlüsselt man nun jeden Buchstaben einzeln, indem einfach der Buchstabe eingesetzt wird, der im Alphabet z Einträge weiter hinten steht. Für $z = 3$, wird der Buchstabe B also beispielsweise mit E verschlüsselt. Ist man am Ende des Alphabets angekommen, wird einfach wieder von vorn begonnen, so dass Z mit C verschlüsselt wird. Insgesamt ergibt sich für $z = 3$ beispielsweise das folgende Schema:



Schreiben Sie eine Funktion `encipher` die ein Wort (nur Großbuchstaben), sowie den Schlüssel z von der Tastatur einliest und die verschlüsselte Version ausgibt. Schreiben Sie außerden eine Funktion `decipher`,

die ein verschlüsseltes Wort mittels Schlüssel z wieder dechiffriert. Einzelne Zeichen können hierbei mit Zahlen $n \in \mathbb{N}$ addiert werden. Den Integer Wert (ASCII) eines Zeichens können Sie in `printf` mittels `%i` ausgeben lassen. Achten Sie darauf, dass das große Alphabet die Werte $65 - 90$ (ASCII) hat. Speichern Sie den Source-Code unter `cesar.c` in das Verzeichnis `serie05`.
Tip: Zur besseren Übersicht ist es sinnvoll die Funktionalität ein einzelnes Zeichen zu chiffrieren in eine eigenständige Funktion auszulagern.

Aufgabe 5.4*. Schreiben Sie eine Funktion `unique`, die einen gegebenen Vektor $v \in \mathbb{N}^m$ sortiert und alle mehrfach auftretenden Einträge entfernt, d.h. zum Beispiel `unique(3 3 1 7 9 7 2) = (1 2 3 7 9)`. Achten Sie darauf, den Speicher entsprechend neu zu allokieren. Schreiben Sie ferner ein aufrufendes Hauptprogramm, welches v von der Tastatur einliesst und `unique(v)` ausgibt. Speichern Sie den Source-Code unter `unique.c` in das Verzeichnis `serie05`.

Aufgabe 5.5. Die Funktion `unique` aus Aufgabe 5.4 ist von der Form `pointer = unique(pointer)`. Ändern Sie die Implementierung auf die Form `void unique(&pointer)` ab. Erklären Sie den Unterschied. Speichern Sie den Source-Code unter `unique2.c` in das Verzeichnis `serie05`.

Aufgabe 5.6. Schreiben Sie eine Funktion `cutoff`, die aus einem gegebenen Vektor $v \in \mathbb{R}^n$ alle Einträge entfernt, die grösser als eine gewisse Schranke c sind. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem v und c von der Tastatur eingelesen werden, und der Ergebnisvektor ausgegeben wird. Das Ergebnis soll wieder in v gespeichert werden. Achten Sie also darauf, den Speicher entsprechend neu zu alliieren. Speichern Sie den Source-Code unter `cutoff.c` in das Verzeichnis `serie05`.

Aufgabe 5.7. Schreiben Sie eine Funktion `Umvm` die eine Matrix-Matrix-Multiplikation mit Matrizen vom Typ `Umatrix` realisiert. Welches Format hat die Zielmatrix? Beweisen Sie Ihre Antwort mathematisch. Speichern Sie den Source-Code unter `Umvm.c` in das Verzeichnis `serie05`.

Aufgabe 5.8. Schreiben Sie einen Strukturdatentyp `cdouble`, in dem Realteil und Imaginärteil einer Zahl $a + bi \in \mathbb{C}$ jeweils als `double` gespeichert werden. Schreiben Sie Funktionen `newCDouble`, `delCDouble` sowie die vier Zugriffsfunktionen `setCDoubleReal`, `getCDoubleReal`, `setCDoubleImag` sowie `getCDoubleImag`. Speichern Sie den Source-Code, aufgeteilt in Header-Datei `cdouble.h` und `cdouble.c`, ins Verzeichnis `serie05`.

Aufgabe 5.9. Schreiben Sie eine kleine Bibliothek für die Arithmetik mit komplexen Zahlen. Orientieren Sie sich hierbei an den Folien 123-124 und verwenden Sie den Datentyp aus Aufgabe 5.8. Schreiben Sie Funktionen für Addition, Subtraktion, Multiplikation und Division von zwei komplexen Zahlen. Schreiben Sie außerden eine Funktion `myabs` die den Betrag

$$|z| = \sqrt{a^2 + b^2}$$

einer komplexen Zahl $z = a + ib \in \mathbb{C}$ berechnet und zurückgibt.

Aufgabe 5.10. Schreiben Sie eine Funktion `doubleInvertString`, die eine gegebene Zeichenkette a erhält und die Zeichenkette selbst in invertierter Reihenfolge nochmals an a anhängt und zurückgibt. Orientieren Sie sich hierbei an der Funktion `stringcopy` aus der Vorlesung. Speichern Sie den Source-Code unter `doubleInvertString.c` in das Verzeichnis `serie05`.