Dirk Praetorius,
Thomas Führer,
Michael Feischl

Wintersemester 2013/14
04.12.2013

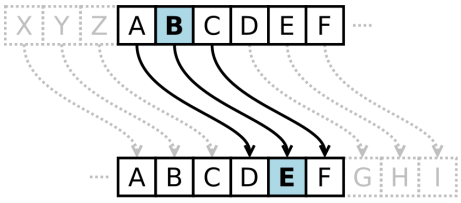
Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 10

Aufgabe 10.1. Schreiben Sie eine Funktion `bin2dec`, die eine im Binärformat gegebene natürliche Zahl $0 \leq z < 256$ in eine Dezimalzahl umrechnet. Die Binärzahl werde als Vektor der Koeffizienten $a_i \in \{0, 1\}$ für $i = 0, \dots, 7$ dargestellt. Dann lässt sich der Wert der Binärzahl mittels $z = \sum_{i=0}^{7} a_i 2^i$ berechnen. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Koeffizienten a_i eingelesen und der Wert von z als Dezimalzahl ausgegeben werden. Speichern Sie den Source-Code unter `bin2dec.c` in das Verzeichnis `serie10`.

Aufgabe 10.2. Schreiben Sie eine `void`-Funktion `dec2bin`, die zu einer natürlichen Zahl $0 \leq z < 256$ die Binärdarstellung berechnet und ausgibt. Es sollen die Koeffizienten $a_i \in \{0, 1\}$ für $i = 0, \dots, 7$ ermittelt werden, sodass $z = \sum_{i=0}^{7} a_i 2^i$ gilt. Anschließend soll die Binärdarstellung in einem geeignetem Format ausgegeben werden. Beispielsweise gebe die Funktion für $z = 77$ die Zeichenfolge `0 1 0 0 1 1 0 1` aus. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem z eingelesen und `dec2bin` aufgerufen werden. Speichern Sie den Source-Code unter `dec2bin.c` in das Verzeichnis `serie10`.

Aufgabe 10.3. 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 außerdem 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 `caesar.c` in das Verzeichnis `serie10`.
Tipp: Zur besseren Übersicht ist es sinnvoll eine eigene Funktion zur Chiffrierung eines einzelnen Buchstabens zu schreiben.

Aufgabe 10.4. Die Verschlüsselung aus Aufgabe 10.3 kann relativ leicht "geknackt" werden. Durch die Buchstabenhäufigkeit — in deutschsprachigen Texten kommt am häufigsten der Buchstabe 'e' vor, am zweithäufigsten der Buchstabe 'n' — kann man auf den Schlüssel $z \in [0, \dots, 25]$ schließen. Die Verschlüsselung aus Aufgabe 10.3 kann verfeinert werden, indem man ein Passwort der Länge $n \in \mathbb{N}$ als Schlüssel benutzt. Dabei wird der erste Buchstabe des Passworts benutzt um den ersten Buchstaben des Textes zu verschlüsseln. Der zweite Buchstabe des Passworts dient zur Verschlüsselung des zweiten Buchstaben des Textes usw. Nach dem letzten Buchstaben des Passworts startet man wieder von vorne, d.h. der $n + 1$-te Buchstabe im Text wird mit dem ersten Zeichen im Passwort verschlüsselt. Als Verschlüsselungsmethode nimmt man dabei für jeden einzelnen Buchstaben wiederum die Cäsar-Chiffre.

Wählt man etwa als Passwort `ACF`, so soll der erste Buchstabe des zu verschlüsselnden Textes um den Wert 0 verschoben werden, der zweite um den Wert 2 und der dritte um den Wert 5. Danach fängt man wieder beim ersten Buchstaben des Passworts an, d.h. der vierte Buchstabe des zu verschlüsselnden Textes wird um den Wert 0 verschoben usw. Implementieren Sie das oben beschriebene Verfahren. Speichern Sie den Source-Code unter `caesarPWD.c` in das Verzeichnis `serie10`.

Aufgabe 10.5. (dynamische Matrizen, effiziente Speicherung) Eine untere Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$

$$L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

hat höchstens $\frac{n(n+1)}{2} = \sum_{j=1}^{n} j$ nicht-triviale Einträge. Erklären Sie wie man auf diese Formel kommt. Schreiben Sie eine Struktur `matrixL`, in der neben der Dimension $n \in \mathbb{N}$ die Koeffizienten L_{jk} in einem dynamischen Vektor ℓ der Länge $\frac{n(n+1)}{2}$ gespeichert werden. Überlegen Sie sich, an welcher Stelle ℓ_m im dynamischen Vektor ein Eintrag L_{jk} gespeichert werden soll. Schreiben Sie die entsprechenden Funktionen zum Allokieren und Freigeben des Speichers einer unteren Dreiecksmatrix (`newMatrixL`, `delMatrixL`) und die Funktion `getMatrixLN`, welche die Größe $n \in \mathbb{N}$ zurückgibt. Implementieren Sie ferner die Zugriffsfunktionen `getMatrixLjk` bzw. `setMatrixLjk`, um auf einzelne Koeffizienten der Matrix zuzugreifen.

Aufgabe 10.6. (effizientes Multiplizieren) Schreiben Sie eine Funktion `matrixvectorL`, die das Matrix-Vektor-Produkt $y = Lx$ mit einer unteren Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ mittels geeigneter Schleifen berechnet. Bei der Berechnung soll auf die offensichtlichen Nulleinträge von L nicht zugegriffen werden, um den Rechenaufwand gering zu halten. Schreiben Sie ferner ein aufrufendes Hauptprogramm in dem die Dimension $n \in \mathbb{N}$ sowie die Einträge der Matrix L und die Koeffizienten des Vektors x eingelesen und Lx ausgegeben werden. Speichern Sie den Source-Code unter `matrixvectorL.c` in das Verzeichnis `serie10`.

Aufgabe 10.7. Schreiben Sie einen Strukturdatentyp `cdouble`, in dem Realteil a und Imaginärteil b einer komplexen Zahl $a+bi \in \mathbb{C}$ jeweils als `double` gespeichert werden. Schreiben Sie Funktionen `cdouble* newCDouble(double a, double b)`, `delCDouble` sowie die vier Zugriffsfunktionen `setCDoubleReal`, `getCDoubleReal`, `setCDoubleImag` sowie `getCDoubleImag`. Speichern Sie den Source-Code, aufgeteilt in Header-Datei `cdouble.h` und `cdouble.c`, in das Verzeichnis `serie10`.

Aufgabe 10.8. Was ist ein Gleitkommazahlsystem? Aus welchen Bestandteilen setzt sich eine Gleitkommazahl zusammen? Wie bestimmt man daraus ihren Wert? Was verbirgt sich hinter den Symbolen `Inf`, `-Inf` und `NaN`? Was ist die Maschinengenauigkeit `eps`? Was ist eine denormalisierte Gleitkommazahl? Was ist ein implizites erstes Bit? Kann es auch zur Basis 7 ein implizites erstes Bit geben?