Dirk Praetorius,
Marcus Page,
Thomas Führer,
Josef Kemetmüller

Sommersemester 2013
25.06.2013

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 14

Die ersten vier Beispiele dieser Übungsserie sollen der Wiederholung von Inhalten der ersten Semesterhälfte dienen.

Aufgabe 14.1. (Strukturen in C) Eine untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ mit

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

hat höchstens $\frac{n(n+1)}{2} = \sum_{j=1}^{n} j$ nicht-triviale Einträge. Schreiben Sie eine Struktur `matrixL`, in der neben der Dimension $n \in \mathbb{N}$ die Koeffizienten L_{ij} in einem dynamischen Vektor der Länge $\frac{n(n+1)}{2}$ gespeichert werden. Schreiben Sie die entsprechenden Zugriffsfunktionen (`newMatrixU`, `delMatrixU`, `getMatrixUDimension`, `getMatrixUij`, `setMatrixUij`), und überlegen Sie sich zuvor, an welcher Stelle l_ℓ im dynamischen Vektor ein Eintrag L_{ij} gespeichert werden soll. Tipp: Speichern Sie L zeilenweise.

Aufgabe 14.2. (Strukturen in C) Das Produkt $L = AB$ zweier unterer Dreiecksmatrizen $A, B \in \mathbb{R}^{n\times n}$ ist wieder eine untere Dreiecksmatrix. Man beweise diese Aussage zunächst mathematisch, indem man sich die Formel für das Matrix-Matrix-Produkt hinschreibe und mittels der Voraussetzung an A und B die Indizes vereinfache. Danach schreibe man eine Funktion `matrixmatrixL`, die die Produktmatrix berechnet und zurückgibt. Dabei sollen natürlich nur die nicht-trivialen Einträge von L, d.h. L_{jk} für $j \geq k$, berechnet werden. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden, d.h. man verwende die anfangs hergeleitete Formel.

Aufgabe 14.3. (Rekursion) Für $n \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}$ definieren wir die Faktorielle durch $f(0) = 1$ und $f(n) := n \cdot f(n-1)$ für $n \geq 1$. Implementieren Sie diese rekursive Funktion und schreiben Sie ein aufrufendes Hauptprogramm, das $n \in \mathbb{N}_0$ einliest und $f(n)$ ausgibt. Schreiben Sie ferner Funktionen

```
nk = binomial(n,k)
```

die den Binomialkoeffizienten $\binom{n}{k}$ berechnen. Dies lässt sich auf verschiedene Weisen realisieren:

- direkt in der Form $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ unter Verwendung der Funktion für die Faktorielle,
- in gekürzter Form $\binom{n}{k} = \frac{n\cdot(n-1)\cdots(n-k+1)}{k\cdot(k-1)\cdots 1}$ mittels geeigneter Schleifen,
- mittels einer rekursiven Funktion, die das Additionstheorem $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ benutzt.

Realisieren Sie alle 3 Varianten und zusätzlich ein Hauptprogramm, das die Zahlen n und k über die Tastatur einliest und den Binomialkoeffizien berechnet und ausgibt.

Aufgabe 14.4. (Strukturen in C) Was tut die folgende Funktion `func` bei Übergabe der Matrix

$$A = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 3 \\ 1 & 2 & 0 & 2 \\ 17 & 4 & 4 & 1 \end{pmatrix}?$$

A sei hierbei in der Struktur aus Serie 7 gespeichert. Geben Sie tabellarisch wieder, welchen Wert die Variablen zum angegebenen Zeitpunkt haben. Welche Funktionalität wird durch `func` bereitgestellt? Was ist an dieser Lösung ineffizient realisiert und wie könnte man das effizienter gestalten?

```
int func(squareMatrix* mat) {
   double foo = 0;
   int mp, dp, tf;
   mp = 1;
   for (dp = 0; dp < getMatrixDim(mat); ++dp) {
      for (tf = dp+1; tf < getMatrixDim(mat); ++tf) {
         foo = getMatrixEntry(mat,dp,tf);
         if ( foo != 0 ) {
            mp = 0;
         }
      /* WERT DER VARIABLEN ZU DIESEM ZEITPUNKT */
      }
   }
   return mp;
}
```

Aufgabe 14.5. (Operatoren,Klassen in C++) Schreiben Sie eine Klasse `Alkohol` zur Speicherung von verschiedenen alkoholischen Getränken. Diese soll folgende Member-Variablen enthalten: Name, Alkoholgehalt in Prozent, Preis in €. Weiters soll für diese Klasse neben einem passenden Konstruktor auch der `operator<` definiert werden. Dieser soll nach besserem Verhältnis $\frac{\text{Vol.\%}}{€}$ bewerten. Definieren Sie auch die Methoden `getName()`, `getPreis()` und `getVolProz()`. Speichern Sie den Source-Code unter `Alkohol.cpp` in das Verzeichnis `serie14`.

Aufgabe 14.6. (Templates in C++) Schreiben Sie eine Template-Funktion `pot(T x, int n)`, welche für einen beliebigen Datentyp `T` (der die Funktionen `operator*` und `operator/` unterstützt) die Funktion x^n realisiert. Testen Sie ihr Beispiel mit verschiedenen Datentypen. Speichern Sie den Source-Code unter `pot.cpp` in das Verzeichnis `serie14`.
Hinweis: Beachten Sie, dass die Funktion auch für negative n zu programmieren ist. Dafür können Sie verwenden, dass `x/x` dem Einselement vom Typ `T` entspricht.

Aufgabe 14.7. (Templates in C++) Schreiben Sie eine Template-Funktion `maximum(std::vector<T> v)`, welche das Maximum eines Vektors von `T`-Objekten zurückliefert. Gehen Sie davon aus, dass die Klasse `T` zumindest die Funktion `operator<` definiert hat. Testen Sie die Funktion für verschiedene Datentypen, davon zumindest einen selbst definierten. (Also eine eigene Klasse, welche den `operator<` definiert.) Speichern Sie den Source-Code unter `maximum.cpp` in das Verzeichnis `serie14`.

Aufgabe 14.8. (Templates in C++) Schreiben Sie eine Template-Funktion `minsort(std::vector<T> v)`, welche einen Vektor von `T`-Objekten übernimmt, aufsteigend sortiert und dann zurückgibt. Gehen Sie davon aus, dass die Klasse `T` zumindest die Funktion `operator<` definiert hat. Das Verfahren soll wie *MinSort* aus der Vorlesung funktionieren. (Siehe Folie 75ff.)
Speichern Sie den Source-Code unter `minsort.cpp` in das Verzeichnis `serie14`.