Dirk Praetorius,
Thomas Führer

Sommersemester 2014
06.06.2014

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 12

Die ersten zwei Beispiele erfordern teilweise, dass Sie sich mit Funktionen oder Containern der Standardbibliothek im Internet vertraut machen. Nutzen Sie dazu eine der beiden Internetseiten:

`http://www.cppreference.com` oder `http://www.cplusplus.com`.

Das letzte Beispiel handelt von Templates, welche nächste Woche in der VO besprochen werden. Auf den Folien aus dem Sommersemester 2013 finden sich Details zu Templates.

Aufgabe 12.1. Ein Paar ist ein C++-Datentyp, der zwei Werte möglicherweise unterschiedlichen Typs enthält. Ein Paar von `double`-Werten wird beispielsweise mit `pair<double,double>` bezeichnet. Ein Paar von Werten wird beispielsweise mithilfe des Aufrufs `pair<double,double>(5.,3.)` erstellt. Um Paare zu verwenden müssen Sie die Header-Datei `map` einbinden!
Schreiben Sie eine Funktion `minmax`, die eine Liste von Gleitkommazahlen erhält, das Minimum und Maximum dieser Liste ermittelt und in Form eines Paares zurückgibt! Geben Sie das Minimum und Maximum in der `main`-Funktion aus! Speichern Sie den Source-Code unter `minmax.cpp` in das Verzeichnis `serie12`.
Hinweis: Auf den ersten Wert eines Paares kann mit `.first` auf den zweiten mit `.second` zugegriffen werden. Beispiel:

```
pair<int, double> x(5, 17.4);
cout << x.first << ", " << x.second << endl; // Ausgabe: 5, 17.4
```

Aufgabe 12.2. Schreiben Sie eine Funktion `sortfile`, die eine Datei zeilenweise in einen Vektor einliest, diesen Vektor lexikografisch sortiert und anschließend ausgibt. Legen Sie eine geeignete Datei an, um Ihr Programm zu testen! Speichern Sie den Source-Code unter `sortfile.cpp` in das Verzeichnis `serie12`.
Hinweis: Folgender Code liest eine Datei zeilenweise ein:

```
fstream datei("Dateiname.txt");
while (datei.good()) {
   string zeile;
   getline(datei, zeile);
}
```

Lösen Sie die Aufgabe, indem Sie diesen Code adaptieren. Binden Sie die Header-Datei `fstream` ein! Hinweis: Der Vergleichsoperator `<` für `strings` entspricht genau der lexikografischen Ordnung. Es gilt also für `strings` `a`, `b` genau dann `a < b`, wenn `a` in der Ausgabe vor `b` kommen soll. Sie können die Funktion `sort` aus der Standardbibliothek verwenden.

Aufgabe 12.3. Implementieren Sie eine Klasse `Person`, welche die Datenfelder `name` und `adresse` enthält. Schreiben Sie auch die zugehörigen Zugriffsfunktionen. Leiten Sie von dieser Klasse eine Klasse `Student` ab, welche die zusätzlichen Datenfelder `matrikelnummer` und `studium` enthält. Leiten Sie von der Klasse `Person` auch eine Klasse `Arbeiter` ab. Erweitern Sie diese Klasse um die Datenfelder `gehalt` und `arbeit`. Überlegen Sie sich auch, welche Zugriffsspezifizierer für welche Datenelemente verwendet werden und begründen Sie dies.

Aufgabe 12.4. Erstellen Sie für die Basisklasse `Person` aus Aufgabe 12.3 eine Methode `void print()`, welche den Namen und die Adresse einer Person am Bildschirm ausgibt. Redefinieren Sie diese Funktion dann jeweils für die Klassen `Student` und `Arbeiter` (Es sollen die zusätzlich definierten Datenelemente auch ausgegeben werden). Schreiben Sie dann noch ein Hauptprogramm, in welchem die `print`-Funktionen der verschiedenen Klassen getestet werden sollen.

Aufgabe 12.5. Wir betrachten die Klasse `Matrix` und die davon abgeleitete Klasse `SquareMatrix` aus der VO. Implementieren Sie für die Klasse `SquareMatrix` die Methode `computeLU`, welche die LU-Zerlegung berechnet. Der Rückgabewert (Matrix $R \in \mathbb{R}^{n\times n}$) sei dabei wieder vom Type `SquareMatrix`, wobei die beiden Dreiecksmatrizen L und U in R gespeichert werden sollen. Die Diagonale von L muss hierbei nicht explizit gespeichert werden. (Warum?)
Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

Wenn aber A eine normalisierte LU-Zerlegung besitzt, so gilt

$$\begin{aligned} u_{ik} &= a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1, \dots, n, \quad k = i, \dots, n, \\ \ell_{ki} &= \frac{1}{u_{ii}} \left(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \right) \quad \text{für } i = 1, \dots, n, \quad k = i+1, \dots, n, \\ \ell_{ii} &= 1 \quad \text{für } i = 1, \dots, n, \end{aligned}$$

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n\times n}$ sind Null.

Aufgabe 12.6. Welchen Aufwand besitzt die LU-Zerlegung aus Aufgabe 12.5? Schreiben Sie das Ergebnis in der $\mathcal{O}$-Notation auf und erklären Sie wie sie auf das Ergebnis gekommen sind.

Aufgabe 12.7. Die Determinante einer Matrix $A \in R^{n\times n}$ kann über die normalisierte LU-Zerlegung aus Aufgabe 12.5 berechnet werden. Es gilt namlich $\det(A) = \det(L)\det(U) = \det(U) = \prod_{j=1}^{n} u_{jj}$. Erweitern Sie die Klasse `SquareMatrix` um eine Methode `detLU`, die die Determinante über die LU-Zerlegung berechnet und zurückgibt. Die Matrix selbst soll hierbei nicht überschrieben werden.

Aufgabe 12.8. 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.) Testen Sie ihre Implementierung anhand von mehreren(!) verschiedenen Typen.
Speichern Sie den Source-Code unter `minsort.cpp` in das Verzeichnis `serie12`.