Dirk Praetorius,
Marcus Page

Sommersemester 2012
16.05.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Achtung: Auf diesem Übungsblatt sind die Aufgaben 1 und 2 freiwillig; sie führen Sie an den Umgang mit Iteratoren und der C++-Standardbibliothek heran und helfen Ihnen dabei, die Pflichtaufgaben 3–6 *besser* zu lösen — sie sind aber für die Bearbeitung der Pflichtaufgaben nicht notwendig.

Die freiwilligen Beispiele erfordern teilweise, dass Sie sich mit Funktionen oder Containern der Standardbibliothek im Internet vertraut machen. Nutzen Sie dazu eine der beiden Internetseiten: `www.cppreference.com` oder `www.cplusplus.com`

Aufgabe 9.1. In C können Sie zu Zeigern Ganzzahlen hinzuaddieren. Verdeutlichen Sie sich dieses Konzept anhand von folgendem C-Code:

```
double foo(double* array, int dim)
{
  double bar = 0.;
  double* begin = array;
  double* end   = array + dim;
  double* ptr;
  for (ptr = begin; ptr != end; ++ptr)
    bar = bar + *ptr;
  return bar / (double) dim;
}
```

Was passiert, wenn man einen Zeiger und eine Ganzzahl addiert? Was berechnet die Funktion `foo`? Beantworten Sie diese Fragen, indem Sie obiges Programm für unterschiedliche Eingaben ausführen!
In C++ verallgemeinern Iteratoren bestimmte Eigenschaften von Zeigern auf C++-Standardcontainer wie `vector` oder `list`. Die Methoden `begin` und `end` liefern analog zu obigem Programmabschnitt Iteratoren auf das erste Element des Containers bzw. auf das erste Element außerhalb des Container zurück. Schreiben Sie eine C++-Version der Funktion `foo`, die anstelle eines C-Arrays einen `vector<double>`-Container erhält und anstelle von Zeigern Iteratoren verwendet. Speichern Sie den Source-Code unter `aufgabe01.cpp` in das Verzeichnis `serie09`.
Hinweis: Ist `v` vom Typ `vector<double>`, so könnten Sie mithilfe von Iteratoren mit

```
vector<double>::iterator it = v.begin();
it = it + 2;
cout << *it << endl;
```

das Element `v[2]` ausgeben.

Aufgabe 9.2. In Aufgabe 1 haben Sie die Funktion `foo` für den Container-Datentyp `vector<double>` implementiert. Schreiben Sie nun eine weitere Variante dieser Funktion, die jedoch als Eingabe anstelle eines Containers vom Typ `vector<double>` einen vom Typ `list<double>` erhält.
Schreiben Sie **noch eine weitere** Variante unter Verwendung der Funktion `accumulate` aus der C++-Standardbibliothek. Binden Sie dazu die Header-Datei `numeric` ein. Speichern Sie den Source-Code unter `aufgabe02.cpp` in das Verzeichnis `serie09`.
Hinweis: Die Funktion

```
double accumulate(list<double>::iterator begin,
                  list<double>::iterator end,
                  double startwert)
```

addiert alle Einträge, die zwischen `begin` und `end` liegen zum Startwert `startwert` hinzu.

Aufgabe 9.3*. Die Frobeniusnorm $\|\cdot\|_F$ einer Matrix $A \in \mathbb{R}^{m\times n}$ definiert durch:

$$\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2}$$

Schreiben Sie zunächst eine Funktion `skalarprodukt`, die das Skalarprodukt zweier Vektoren berechnet **oder** machen Sie sich mit der C++-Funktion `inner_product` vertraut. Schreiben Sie dann eine Funktion `frobeniusnorm`, die die Frobeniusnorm einer Matrix A vom C++-Typ `vector< vector<double> >` unter Ausnützung der Funktionen `skalarprodukt` bzw. `inner_product` berechnet. Speichern Sie den Source-Code unter `frobeniusnorm.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.4*. Der Verschlüsselungsalgorithmus ROT13 ersetzt jeden Buchstaben der Eingabe zyklisch durch den im Alphabet um 13 Buchstaben weiter hinten (oder vorne) stehenden Buchstaben. Die Buchstaben A, B, C, ... werden auf N, O, P, ... abgebildet, die Buchstaben N, O, P, ... auf A, B, C, ...
Setzen Sie diesen Algorithmus in der Funktion `rot13` um. Ihre Funktion braucht nur für Großbuchstaben zu funktionieren.
Schreiben Sie nun eine weitere Funktion `rot_n`, der Sie zusätzlich übergeben können, um wieviele Stellen die Eingabe rotiert werden soll. Überlegen Sie sich, wie man mit `rot_n` verschlüsselte Texte entschlüsseln kann. Schreiben Sie eine Funktion `unrot_n`, die intern die Funktion `rot_n` verwendet.
Hinweis: Ein Zeichen ist vom Typ `char` und wird intern durch eine Zahl dargestellt, 'A' beispielsweise durch 65. Ziehen Sie von einem Buchstaben 'A', ..., 'Z' den Buchstaben 'A' ab, so ist das Ergebnis intern eine Zahl zwischen 0 und 25. Führen Sie dort die notwendigen Rechenoperationen durch und addieren Sie anschließend wieder den Buchstaben 'A' hinzu! Weitere Informationen unter ASCII auf Wikipedia.
Fleißaufgabe: Implementieren Sie die Funktion `rot13` mithilfe der C++-Funktion `transform`. Welche Schwierigkeiten treten bei der Implementierung der Funktion `rot_n` mithilfe von `transform` auf?

Aufgabe 9.5*. Schreiben Sie eine Funktion `sortfile`, die eine Datei zeilenweise in einen Vektor einliest, diesen Vektor lexikografisch sortiert und anschließend ausgibt. Der Name der Datei soll als Kommandozeilenparameter übergeben werden können. Legen Sie eine geeignete Datei an, um Ihr Programm zu testen! Speichern Sie den Source-Code unter `sortfile.cpp` in das Verzeichnis `serie09`.
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 `string`s entspricht genau der lexikografischen Ordnung. Es gilt also für `string`s `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 9.6*. 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 `serie09`.
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 9.7. Tag-Clouds stellen die Wichtigkeit von Begriffen in Texten dar, indem sie diese entsprechend ihrer Häufigkeit unterschiedlich groß darstellen. Häufig eingesetzt werden sie in sozialen Medien und Netzwerken. In den folgenden Aufgaben implementieren Sie eine Tag-Cloud. Machen Sie sich dazu mit dem `map`-Container vertraut.
Schreiben Sie eine Funktion `worthaeufigkeit`, der Sie einen Dateinamen übergeben können. Die Funktion soll einen Container vom Typ `map<string,int>` zurückliefern, der alle in der Datei vorkommenden Wörter mit zugehörigen Häufigkeiten enthält.
Anleitung: Für jedes eingelesene Wort überprüfen Sie mit der `find`-Methode, ob es sich bereits in Ihrem `map`-Container befindet. Ist das der Fall, so liefert die Funktion einen Iterator — nennen wir ihn `it` — zurück und Sie können die Häufigkeit mittels `it->second += 1` erhöhen. Andernfalls liefert die `find`-Methode den `end`-Iterator zurück und Sie können mittels `insert` ein passendes Werte-Paar einfügen.

Aufgabe 9.8. Ihre in Aufgabe 9.7 geschriebene Funktion berechnet nun zwar wie häufig jedes Wort vorkommt, aber Sie kennen die häufigsten Wörter noch nicht. Schreiben Sie eine Funktion `sortiere`, der Sie den in Aufgabe 9.7 berechneten `map<string,int>`-Container übergeben und die einen Container vom Typ `vector<pair<string,int>>` zurückgibt, der die einzelnen Wörter mit zugehörigen Häufigkeiten aufsteigend sortiert nach ihren Häufigkeiten enthält.
Anleitung: Verwenden Sie `sort` aus der Standardbibliothek, sowie die Funktion

```
bool less_than(pair<string,int> a, pair<string,int> b) {
  return a.second < b.second;
}
```

als Ordnungsfunktion.

Aufgabe 9.9. Unter den häufigsten Wörtern finden sich auch Bindewörter oder Artikel. Schreiben Sie eine Funktion `entferne_bindewoerter`, der Sie zwei Parameter übergeben: den Container vom Typ `vector<pair<string,int> >`, den Sie in Aufgabe 9.8 erzeugt haben und den Namen einer Datei, die eine Liste mit Bindewörtern enthält. Die Funktion soll die Datei wortweise einlesen und alle Einträge aus dem Vektor streichen, deren erster Eintrag gleich einem der eingelesenen Wörter ist. Der veränderte Vektor soll zurückgegeben werden.
Hinweis: Alternativ zu einer naiven Implementierung können Sie `remove_if` und eine passende Prädikatsfunktion verwenden.

Aufgabe 9.10. Implementieren Sie die Funktionen `unique` und `checkPalindrom` aus der vorangehenden Übung mithilfe der Standardbibliothek.
Hinweis: Verwenden Sie die Funktionen `sort` und `unique` um `unique` zu realisieren. Verwenden Sie `equal` und einen reverse iterator (siehe `rbegin`) um `checkPalindrom` zu realisieren.