Universität des Saarlandes
FR 6.2 Informatik
Prof. Dr. Hans-Peter Lenhof
Dipl. Inform. Andreas Hildebrandt

Programmierung II, SS 2003
Musterlösung zur 1. Übung

Hinweis: Wie in der Vorlesung angekündigt, gehen die ersten drei Übungen nicht in die Wertung ein!

Zum Kompilieren der Musterlösungen muss sich eine Datei mit Namen **Makefile** im selben Verzeichnis wie der Quellcode befinden, die folgenden Inhalt hat:

```
aufgabe1: hallobremser.cpp
      g++ hallobremser.cpp -o hallobremser -g -Wall -pedantic

aufgabe2: vectorsort.cpp
      g++ vectorsort.cpp -o vectorsort -g -Wall -pedantic

aufgabe3: binarysearch.cpp
      g++ binarysearch.cpp -o binarysearch -g -Wall -pedantic
```

Mit folgenden Aufrufen kann man damit die Musterlösungen kompilieren:

- `make aufgabe1` Musterlösung zur Aufgabe 1 wird kompiliert.
- `make aufgabe2` Musterlösung zur Aufgabe 2 wird kompiliert.
- `make aufgabe3` Musterlösung zur Aufgabe 3 wird kompiliert.

Aufgabe 1: Erste Schritte...
Der folgende Code befindet sich in der Datei **hallobremser.cpp**

```
#include <iostream>

using namespace std;

int main(int argc, char** argv)
{
  cout << "Lieber Bremser, ich wäre gern in Ihrer Übungsgruppe!" << endl;

  return 0;
}
```

Sie können das erzeugte Progamm mittels

```
./hallobremser > hallo
```

aufrufen und die Ausgabe in die Datei **hallo** umleiten. In der erzeugten Datei sollte sich folgend Zeile befinden:

```
Lieber Bremser, ich wäre gern in Ihrer Übungsgruppe!
```

Aufgabe 2: Der `vector<>`-Datentyp
Der folgender Code befindet sich in der Datei **vectorsort.cpp**

```
#include <iostream>   // brauchen wir für cin, cout
#include <vector>     // brauchen wir für den vector
#include <algorithm>  // brauchen wir für sort

using namespace std;

int main(int argc, char **argv)
{
  vector<int> daten; // wir belegen keinen Speicher vor, denn
                     // wir wissen ja noch nicht, wie viel wir
                     // einlesen müssen

  int zahl;          // speichert die Eingabe

  cin >> zahl;

  /**************************************************
   *  Die while - Schleife führt den Code zwischen
   *  { und } so lange aus, wie die Bedingung zwischen
   *  ( und ) erfüllt ist!
   **************************************************/
  while (zahl != 42)
  {
    daten.push_back(zahl); // hängt "zahl" an den vector an,
                           // wobei, wenn nötig, Platz dafür geschaffen
                           // wird. Wir bekommen allerdings ein Problem,
                           // falls der Speicher dafür nicht mehr ausreicht.
                           // Wie man solche Fälle abfängt, lernen wir in
                           // einer späteren Übung.

    cin >> zahl;           // liest die nächste Zahl ein
  }

  // So, genug gelesen. Wir können jetzt zum Sortieren schreiten
  sort(daten.begin(), daten.end());      // sortiert den ganzen vector

  // Und schließlich noch das Ergebnis ausgeben
  for (unsigned int i=0; i<daten.size(); i++)
```

```
  {
    cout << i << " " << daten[i] << endl;
  }

  // So. Genug für heute.
  return 0;
}
```

Aufgabe 3: Binäre Suche
Der folgender Code befindet sich in der Datei **binarysearch.cpp**

```
#include <vector>
#include <math.h>
#include <iostream>

using namespace std;

/*************************************************************
 * Diese Funktion implementiert die binäre Suche.
 * Zurückgegeben wird der Index des gesuchten Elementes.
 * Eingabeparameter sind:
 *  - eine const-Referenz auf den zu durchsuchenden Vektor
 *  - die zu suchende Zahl
 *  - die untere Suchgrenze
 *  - die obere  Suchgrenze
 *
 * Wichtig: die Verwendung einer Referenz statt eines
 *  "kompletten" Vektors verhindert das andauernde hin-
 *  und herkopieren
 *************************************************************/
int binarysearch(const vector<int>& daten, int gesucht, int von, int bis)
{
  // auch sowas kommt vor...
  if (bis < von)
  {
    // dann gibt's das Element auch nicht
    return -1;
  }

  // vielleicht sind wir ja schon fertig?
  if (von == bis)
  {
    // entweder wir haben das Element gefunden...
    if (daten[von] == gesucht)
    {
      return von;
```

```
    }
    else // oder es kommt nicht vor.
    {
      return -1;
    }
  }

  // ansonsten: sehen wir uns die Mitte des zu durchsuchenden
  //            Bereichs mal an...

  int mitte = (int) floor((bis - von) / 2.) + von;
  int daten_mitte = daten[mitte];

  if (daten_mitte == gesucht)        // wir sind fertig...
  {
    return mitte;
  }
  else
  {
    if (daten_mitte < gesucht)  // die untere Hälfte ist uninteressant
    {
      return (binarysearch(daten, gesucht, mitte+1, bis));
    }
    else                       // die obere Hälfte ist uninteressant
    {
      return (binarysearch(daten, gesucht, von, mitte-1));
    }
  }
}

int main(int argc, char **argv)
{
  // Wir lassen uns über die Kommandozeile angeben, welches
  // Element wir in dem Vektor suchen sollen.
  // argv[0] ist immer der Name des aufgerufenen Programmes,
  // argv[1] das erste Kommandozeilenargument
  if (argc != 2)
  {
    cout << "Aufrufen mit " << argv[0] << " " << "gesuchtes_element" << endl;

    return 1;
  }

  // Das Argument ist eine Zeichenkette... Wir können diese mit
  // atoi aus der stdlib in eine Zahl verwandeln
  int element = atoi(argv[1]);

  // zuerst erstellen wir mal unseren Vektor...
```

```
  // Als kleinen Test werden wir mal einen Vektor von
  // 100 Elementen durchsuchen, also geben wir diese Größe
  // beim Konstruieren gleich mit an
  vector<int> daten(100);

  // diesen Vektor wollen wir jetzt mit den Zahlen von
  // 0 bis 99 füllen
  for (int i=0; i<100; i++)
  {
    daten[i] = i;
  }

  // Nun suchen wir nach dem Element in unserem Vektor
  int position = binarysearch(daten, element, 0, 99);

  cout << element << " " << position << endl;

  if ((position >= 0) && (element == position))
  {
    cout << "Hat funktioniert!" << endl;
  }
  else
  {
    cout << "Hupps! Da stimmt wohl noch was nicht...!" << endl;

    return 1;
  }

  // so, das war's
  return 0;
}
```
