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

1. Übung zur Vorlesung Programmierung II, SS 2003

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

Aufgabe 1: Erste Schritte...
Machen Sie Sich mit ein wenig mit Ihrem Account im CIP-Pool vertraut. Suchen Sie Sich dazu einen Texteditor Ihrer Wahl aus und erstellen Sie damit das folgende Programm:

```
#include <iostream>

using namespace std;

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

  return 0;
}
```

Speichern Sie dieses Programm als hallobremser.cpp, compilieren Sie es (wobei Sie die evtl. auftretenden Warnungen ausnahmsweise ignorieren dürfen...) und führen Sie es aus. Leiten Sie die Ausgabe des Programms in die Datei `aufgabe1` um, und schicken Sie diese Datei Ihrem Übungsgruppenleiter per email.

Aufgabe 2: Der `vector<>`-Datentyp
Der Datentyp `vector<>` ist ein „komfortabler“ Array-Datentyp aus der C++ Standard Template Library (STL), den Sie in der Vorlesung noch genauer kennen lernen werden. In dieser Aufgabe wollen wir Sie mit den grundlegenden Funktionen dieser Datenstruktur vertraut machen. Den Typ der zu speichernden Elemente geben Sie bei der Deklaration des `vector<>` zwischen den Klammern mit an. So erstellt z.B. `vector<int> v;` ein „Array“ aus `int` Zahlen.
Eine Online-Referenz finden Sie unter `http://www.sgi.com/tech/stl/Vector.html`, und einige der wichtigsten Funktionen können Sie diesem kleinen Beispiel entnehmen:

```
#include <vector>    // liefert die Definition von vector aus der STL
#include <algorithm> // liefert die Definition von   sort aus der STL
#include <iostream>  // liefert die Definition von   cout aus der STL

using namespace std;
```

```
int main(int argc, char** argv)
{
  vector<int> v(100); // Belegt Speicher für 100 Elemente

  for (int i = 0; i < v.size(); i++) // v.size() gibt die Elementanzahl zurück
  {
    v[i] = i;
  }

  v[42] = 4711; // Belegt das 43. Element (Zählung beginnt bei 0)
                // mit der Zahl 4711

  v.push_back(815); // Vergrößert v auf 101 Elemente, und
                    // belegt v[100] mit der Zahl 815

  sort(v.begin(), v.end()); // Sortiert alle Elemente zwischen v.begin()
                            // (dem Anfang von v) und v.end() (dem Ende von v)

  for (int i = 0; i < v.size(); i++)
  {
    cout << i << " " << v[i] << endl;
  }

  return 0;
}
```

Schreiben Sie ein Programm, welches so lange Zahlen von der Eingabeaufforderung in einen `vector<int>` einliest, bis die Zahl 42 eingegeben wird. Sortieren Sie anschließend diesen `vector` und geben Sie das Ergebnis aus.

Aufgabe 3: Binäre Suche

Einen schon sortierten `vector` v der Länge N, $v = (v[0], \cdots, v[N-1])$ kann man besonders einfach nach einem *darin enthaltenen* Element e durchsuchen:
Betrachte dazu die Mitte $m = N/2$ des Vektors.

- falls $v[m] = e$, sind wir fertig
- falls $v[m] > e$, dann muß gelten $e \in \{v[0], \cdots, v[m-1]\}$
  $\Rightarrow$ durchsuche statt v nur den interessanten Bereich $(v[0], \cdots, v[m-1])$
- falls $v[m] < e$, dann muß gelten $e \in \{v[m+1], \cdots, v[N-1]\}$
  $\Rightarrow$ durchsuche statt v nur den interessanten Bereich $(v[m+1], \cdots, v[N-1])$

Iteration dieses Schemas liefert eindeutig das gesuchte Element, falls es in v enthalten ist (wie können Sie den Fall $e \notin v$ abfangen?). Dieser Algorithmus wird als *binäre Suche* bezeichnet.
Implementieren Sie eine solche binäre Suche für Arrays vom Typ `vector<int>`.
