ETH Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut für theoretische Informatik
Dr. B. Gärtner

21. Dezember 2004

Informatik für Mathematiker und Physiker Serie 10 WS 04/05

URL: `http://www.ti.inf.ethz.ch/ew/courses/Info1_04/`

Aufgabe 1 (8 Punkte)

Schreiben Sie ein Programm, das Bubblesort und Quicksort visualisiert. Verwenden Sie hierbei folgende Typen:

```
typedef std::pair<unsigned int,unsigned int>  Pair;
typedef std::vector<Pair>                     Vec;
typedef Vec::iterator                         It;
```

Der Typ `std::pair<T1,T2>` ist in der Bibliothek `<utility>` definiert. Er repräsentiert ein geordnetes Paar, dessen erste Komponente vom Typ `T1` und dessen zweite Komponente vom Typ `T2` ist. Für eine Variable `t` vom Typ `std::pair<T1,T2>` bezeichnet man mit `t.first` die erste Komponente und mit `t.second` die zweite Komponente. Auf Paaren sind die gewohnten Ordnungsrelationen <, <=, >, etc. definiert, und zwar als lexikographische Operationen: Ausschlaggebend ist zunächst die erste Komponente; nur im Fall, dass hier Gleichheit herrscht, wird in der zweiten Komponente verglichen.

In dieser Übung sollen Sie einen Vektor von Zahlen sortieren. Diese Zahlen werden in der ersten Komponente der Einträge eines Vektors `v` vom Typ `Vec` repräsentiert. Die zweite Komponente eines jeden Eintrags gibt dessen Position im Vektor an; d.h. es gilt immer `v[i].second==i`, für alle $0 \leq$ `i` $<$ `v.size()`. Die explizite Darstellung dieser Information erleichtert die im folgenden beschriebene graphische Ausgabe.

Graphische Ausgabe: Ein Vektor ganzer Zahlen kann als sogenanntes Balkendiagramm graphisch dargestellt werden: Für jeden Wert `v[i]` zeichnet man in Spalte `i` (`==v[i].second`) eine Linie der Höhe `v[i].first`, d.h. eine Linie $(\texttt{i}, 0) \rightarrow (\texttt{i}, \texttt{v[i].first})$.

Erstellen Sie eine Funktion `draw_segment(int x, int y)`, die in Spalte `x` eine Linie der Höhe `y` zeichnet. Desweiteren schreiben Sie eine Funktion `graphic_swap(It a, It b)`, welche sowohl die Einträge `(*a).first` und `(*b).first` als auch die entsprechenden Balken im Balkendiagramm vertauscht.

Der Programmablauf gliedert sich wie folgt.

Eingabe: Lesen Sie zu Beginn des Programms einen Initialwert für den Zufallszahlengenerator (siehe Serie 5) ein und fragen Sie, ob Quicksort oder Bubblesort verwendet werden soll.

Initialisierung: Dann fügen Sie in `v` die Zahlen 0 bis 511 ein. (Genauer gesagt die Paare (i,i), für $0 \leq$ i ≤ 511.) Dabei zeichnen Sie auch das entsprechende Balkendiagramm ins Fenster `ifm::wio`. Nachdem der initialisierte Vektor einmal im Fenster dargestellt wurde, sollte er nur noch mittels der Funktion `graphic_swap` verändert werden. So ist sichergestellt, dass das Balkendiagramm korrekt ist.

Permutation: Im nächsten Schritt permutieren Sie `v` zufällig (siehe Serie 9).

Sortieren: Je nach Benutzereingabe wird `v` entweder per Quicksort oder Bubblesort sortiert.

Ende: Am Programmende sollte auf einen Mausclick ins Fenster gewartet werden, damit das Fenster nicht unvermittelt geschlossen wird.

Zur Lösung der Aufgabe verwenden Sie die Funktionen aus der Vorlesung bzw. den vorangegangenen Übungen sowie die Graphik-Bibliothek `<IFM/window>`, die Sie im Rahmen von Serie 5 installiert haben.

Abgabe: Aufgabe 1: bis 17. Januar 2005, 16.00 Uhr, per Email.
In der nächsten Vorlesungswoche (10.–14.01.2005) wird es eine Schnellübung geben!

Fiona Wiemacher.

Eidgenössische Technische Hochschule Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut für theoretische Informatik
Dr. B. Gärtner

21. Dezember 2004

Informatik I: Material aus der Vorlesung

Programm: quicksort.C

```
// Programm: quicksort.C
// Sortieren eines Vektors von Zahlen mittels Quicksort.

#include <iostream>
#include <cassert>
#include <vector>
#include <algorithm>
#include <IFM/math.h>

typedef std::vector<unsigned int>  Vec;
typedef Vec::iterator              It;

namespace ifm {

  It split(It b, It e)
  // PRE: [b,e) ist ein nichtleerer gueltiger range.
  // POST: Die Elemente in [b,e) wurden permutiert.
  //   Rueckgabewert ist ein s aus [b,e), so dass
  //   fuer alle i in [b,s): v[i] <= pivot,
  //   v[s] == pivot sowie
  //   fuer alle i in (s,e): v[i] > pivot,
  //   wobei pivot == *b beim Aufruf der Funktion.
  {
    assert(b != e);
    It i = b;
    for (It j = ++i; j != e; ++j)
      // Invariante: *[b,i) <= *b && *[i,j) > *b.
      if (*j <= *b) std::swap(*(i++), *j);

    // Tausche das Pivot-Element in die richtige Position
    // (am Ende des ersten Blocks)
    std::swap(*b, *--i);
    return i;
  } // split(b, e)

  void quicksort(It b, It e)
  // PRE: [b,e) ist ein gueltiger range.
  // POST: Die Elemente in [b,e) sind aufsteigend sortiert.
  {
    if (b == e) return;
    It mid = ifm::split(b, e);
    ifm::quicksort(b, mid);
    ifm::quicksort(++mid, e);
  } // quicksort(b, e)

  void random_shuffle(It b, It e)
  // POST: [b,e) ist zufaellig permutiert. (d.h. die (Multi-)Menge
  // der Elemente ist unveraendert, aber die Reihenfolge ist
  // zufaellig gleichverteilt unter allen moeglichen Reihenfolgen.)
  {
    double rand = ifm::random(0);
    for (; b != e; ++b) {
      std::iter_swap(b, b + int((e-b) * rand));
      rand = ifm::random(rand);
    }
  }

} // namespace ifm

int main()
{
  // Initialisierung
  const unsigned int n = 128;
  Vec v;
  for (unsigned int i = 0; i < n; ++i)
    v.push_back(i);

  // Zufaellig permutieren
  ifm::random_shuffle(v.begin(), v.end());
  for (It i = v.begin(); i != v.end(); ++i)
    std::cout << *i << " ";
  std::cout << std::endl;

  // Sortieren
  ifm::quicksort(v.begin(), v.end());
  for (It i = v.begin(); i != v.end(); ++i)
    std::cout << *i << " ";
  std::cout << std::endl;

  return 0;
}
```

Programm: bubblesort.h

```
// Programm: bubblesort.h
// Sortieren eines Vektors von Zahlen mittels Bubblesort.

namespace ifm {

  void bubble_sort(It b, It e)
  // PRE: [b,e) ist ein gueltiger range.
  // POST: Die Elemente in [b,e) sind aufsteigend sortiert.
  {
    // Wurden im aktuellen Durchlauf zwei Elemente getauscht?
    bool swapped = true;
    while (b != e && swapped) {
      swapped = false;
      It prev = b;
      It cur = prev;
      while (++cur != e) {
        if (*cur < *prev) {
          std::swap(*cur, *prev);
          swapped = true;
        }
        prev = cur;
      }
      // Das letzte Element ist jetzt auf jeden Fall korrekt:
      --e;
    }
  } // bubble_sort(b, e)

} // namespace ifm
```