BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

*Fachbereich 7*

MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
*Praktische Informatik / Numerik*

e-mail: Juergen.Buhl@math.uni-wuppertal.de

# Algorithmen und Datenstrukturen (Informatik III)

WS1999/2000 – Übungsblatt 10

Abgabetermin: 2. Februar 2000

**Aufgabe 1.** *Einfach gekettete Listen*

Testen Sie die folgende Implementierung einer einfach geketteten Liste:

```
////////////////////////////////
// Datei:   linkedList.cc
// Version: 1.0
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
////////////////////////////////

#include <iostream>
#include <string>
#include <assert.h>

#include <stdlib.h>

using namespace std;

template<class T>
class linkedList{

  struct Knoten{
    T Daten;
    Knoten *next;
    Knoten(const T& daten, Knoten *p):
        Daten(daten), next(p) {};
  } *Anfang;

public:

  void clear_all(Knoten* abStelle){
    if (abStelle != 0) {
```

```
      clear_all(abStelle->next);
      delete abStelle;
    };
  };
  // danger: dangling pointer at end of restlist, if not used
  //         at starting position of list

public:

  linkedList(): Anfang(0) {};

  ~linkedList() { clear_all(Anfang); };

  void push_front(const T& daten){
    Knoten* temp = new Knoten(daten, Anfang);
    assert(temp != 0);
    Anfang = temp;
  };

  void pop_front() {
    if (Anfang != 0) {
      Knoten* temp(Anfang);
      Anfang = Anfang->next;
      delete temp;
    };

  }

  class iterator;
  friend class iterator;

  class iterator {
    Knoten* aktuell;
  public:
    iterator(Knoten* init = 0): aktuell(init) {};
    T& operator*() { return aktuell->Daten; };
    const T& operator*() const {return aktuell->Daten; };
    iterator& operator++(){ // praefix
      if (aktuell != 0) aktuell = aktuell->next;
      return *this;
    };
    iterator operator++(int){ // postfix
      iterator temp = *this;
      ++*this;
      return temp;
    };
    bool operator!=(const iterator& x) const {
      return (aktuell != x.aktuell);
    };
```

```
  };

  iterator begin() const { return iterator(Anfang); };
  iterator end() const   { return iterator(); };
};

int main()
{

  linkedList<string> stringList;
  stringList.push_front("Erster eingefuegter Listenknoten");
  stringList.push_front("Zweiter eingefuegter Listenknoten");
  //stringList.pop_front();
  stringList.push_front("Dritter eingefuegter Listernknoten");
  // ...
  stringList.push_front("Vierter eingefuegter Listernknoten");
  stringList.push_front("ENDE!");

  linkedList<string>::iterator pos(stringList.begin());
  for (; pos != stringList.end(); pos++)
    cout << *pos << endl;

  linkedList<string>::iterator pos2(stringList.begin());
  pos2++; pos2++; pos2++;
  cout << endl;

  pos=pos2;
  for (; pos != stringList.end(); pos++)
    cout << *pos << endl;

 return 0;
}
```

Ergänzen Sie eine Methode `void clear()`, die die ganze Liste löschen soll; vergessen Sie weder `delete` noch eine sinnvolle Wertzuweisung an das private Attribut `Anfang`.
Konzipieren Sie eine Methode `void erase_from(iterator p)`, die das Listenendstück, das an der Stelle `p` beginnt, löschen soll. Programmieren und testen Sie (auch die Extremfälle). Vergessen Sie nicht, die `next`-Referenz der Vorgängerstelle auf den Wert `0` zu setzen; finden sie diese einmal durch Suchen ab `Anfang`, ein anderes Mal durch Benutzung eines modifizierten Iterators, der zwei private Attribute `aktuell, vorgaenger`, einen Observator für `vorgaenger`, ... enthalten soll. Welche Vor- bzw. Nachteile haben die beiden Methoden?

**Aufgabe 2.** *Doppelt gekettete Listen*

Ändern sie die Liste aus Aufgabe 1 mit allen ihren Methoden in der Implementierung auf eine doppelt gekettete Knotenstruktur ab: `next, before`.

Implementieren und testen Sie. Welche Methoden können jetzt (in der Laufzeit) günstiger realisiert werden? Führen Sie diese Änderungen durch und testen Sie.
Implementieren Sie den Dekrement-Operator `--`. Warum ist dieser bei einfach geketteten Listen nicht sinnvoll?

**Aufgabe 3.** *Rekursionen in einfach geketteten Listen*

Schreiben Sie (rekursive) Funktionen, die die Werte aller Listen-Datenfelder einmal in der Reihenfolge vom Beginn bis zum Ende der Liste und ein anderes Mal in der Reihenfolge vom Ende bis zum Beginn ausdruckt.
Erzeugen Sie eine 15 Knoten lange Liste mit int-Datenfeldern.
Schreiben Sie eine (rekursive) Funktion, die die Summe des ersten bis i-ten Listendatenfeldes für $i = 1...m$ ausdruckt ($m$ Zeilen Ausdruck), wobei $m$ die Länge der Liste sein soll.
Schreiben Sie analog eine (rekursive) Funktion, die die Summe des m-ten bis i-ten Listendatenfeldes für $i = m...1$ ausdruckt ($m$ Zeilen Ausdruck), wobei $m$ ebenfalls die Länge der Liste sein soll.

**Aufgabe 4.** *STL*

Die STL stellt eine generische Klasse `list` zur Verfügung:

```
////////////////////////////////
// Datei:   STLlist.cc
// Version: 1.0
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
////////////////////////////////

#include <iostream>
#include <string>
#include <list>

using namespace std;

int main()
{
 {
  list<string> stringList;
  stringList.push_front("Erster eingefuegter Listenknoten");
  stringList.push_front("Zweiter eingefuegter Listenknoten");
  // ...
  stringList.push_front("ENDE!");

  list<string>::iterator pos(stringList.begin());
  for (; pos != stringList.end(); pos++)
    cout << *pos << endl;
 };
 {
  typedef list<double> rliste;
```

```
  rliste rl;
  rl.push_front(1.1);
  rl.push_front(1.2);
  rl.push_front(1.3);
  rl.push_front(1.4);
  rl.push_front(1.5);
  rliste::iterator pos(rl.begin());
  for(; pos != rl.end(); pos++)
    cout << *pos << endl;
 };

  return 0;
}
```

Testen Sie dies Beispielprogramm.
Informieren Sie sich mittels

file:/opt/share/SUNWspro/DOC5.0/lib/locale/C/html/index.html

über die „Standard C++ Library 2.0“ und insbesondere über die Klasse `list`:

file:/opt/share/SUNWspro/DOC5.0/lib/locale/C/html/stdlib/stdref/lis_3222.htm

Stellen Sie für `list` eine Übersicht über die vorhandenen Attribute und Methoden mit je einem Kurzanwendungsbeispiel zusammen.

---

Date: Mon, 17 Jan 2000 15:11:39 +0000
From: Brian Randell <Brian.Randell@newcastle.ac.uk>
Subject: Bug lists babies as aged 100

Thousands of newborn babies have been listed officially as 100 years old. Computers at English register offices are refusing to recognise the year as 2000 and are printing 1900 on birth certificates. [...]
[Source: The (London) *Times*, 17 Jan 2000; The full story is online at:

http://www.the-times.co.uk/news/pages/tim/2000/01/17/timnwsnws01034.html?1069542

]
Brian Randell, University of Newcastle, Newcastle upon Tyne, NE1 7RU, UK
+44 191 222 7923 http://www.cs.ncl.ac.uk/~brian.randell/

---

Date: Mon, 17 Jan 2000 10:58:36 +0000
From: David H Smith <david.h.smith@gecm.com>
Subject: Y2K Problems with Flight Sim 2000 Professional Edition?

I managed to get Microsoft Flight Sim 2000 Pro Edition for Xmas, great! After installing I went to the Microsoft web site and found an update - of course there was one - 9 megabytes in total. I downloaded it, installed it, everything was fine.

A few days later I did my in-frequent disk cleanups, etc. I had not run scandisk for ages so set it off. I was surprised when it announced that it had found a bad file. The file was with one of the Flight Sim 2000 files, and the problem was that it had an invalid date. This problem occurred with all the Flight Sim 2000 files that had come as part of the update I had downloaded.

Was it a Y2K problem? I'm not sure. Everything worked okay and McAfee Virus Checker didn't complain about funny dates on the files. Of course, it could have been a problem with the disk scan program.

Dave Smith

---