10.4 Templates und die STL/Containertyp list und Iteratoren

10.4.1 Problem

In C und vielen anderen Programmiersprachen gibt es die Möglichkeit, mit dynamischem Speicher eine *einfach verkettete lineare Liste* aufzubauen (Abbildung 1). Jedes Element einer solchen Liste

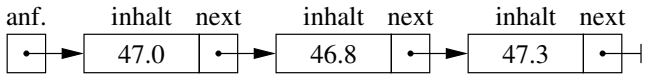

Abbildung 1: Einfach verkettete lineare Liste

besteht aus einer Nutzlast (hier `inhalt` genannt) und einem Zeiger auf das nächste Element (hier `next`) genannt zum Verbinden der einzelnen Elemente:

```
1 \struct element_t{
2    double inhalt;
3    struct element_t *next;
4 };
```

Beim letzten Element in der Liste ist `next` gleich `NULL`. Im Quelltext des Programms braucht man nur eine einzige Variablen (hier `anf` genannt, die ein Zeiger auf das erste Element ist.

Der Vorteil einer linearen Liste ist, dass man schnell Elemente am Ende hinzufügen kann, genauso schnell aber auch am Anfang oder in der Mitte. Dazu brauchen keine Elemente verschoben werden; es reicht, zwei Zeiger richtig zu setzen (siehe Abbildung 2). Und wenn man nacheinander

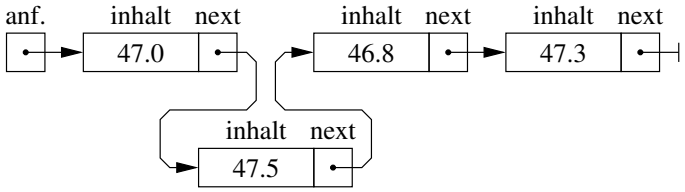

Abbildung 2: Hinzufügen eines Elementes zu einer linearen Liste

auf alle Elemente einer linearen Liste zugreifen will, geht das auch genauso schnell wie bei einem Array.

Ein Nachteil einer linearen Liste gegenüber dem Array liegt darin, dass man zum Zugriff auf ein beliebiges Element die Liste so lange durchlaufen muss, bis man es erreicht hat.

Der Zugriff am Ende der Liste gelingt schneller, wenn man eine *doppelt verkettete Liste* verwendet (Abbildung 3). Hier zeigt von jedem Element auch ein Zeiger `prev` zum vorherigen Element:

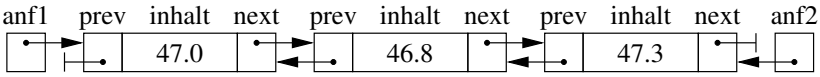

Abbildung 3: Doppelt verkettete lineare Liste

```
1 \struct element_t{
2    struct element_t *prev;
3    double inhalt;
4    struct element_t *next;
5 };
```

Diese Liste hat praktisch zwei Anfänge `anf1` und `anf2`, die beide im Quelltext angelegt sein müssen.

Ein Nachteil für den Programmierer in C liegt darin, dass er die Logik für das Hinzufügen, das Zugreifen und das Löschen selbst programmieren muss. Selbstverständlich gibt es in C Bibliotheken (wie GLib), die Listen im Angebot haben; nur leider sind sie nicht in der Standard-Bibliothen enthalten.

10.4.2 Lösung

In C++ bietet die STL auch für Listen eine Lösung an. Es handelt sich komfortablerweise um eine *doppelt verkettete Liste*.

liste1.cpp

```
1  #include <iostream>
2  #include <list>
3  using namespace std;
4
5  int main(void)
6  {
7      list <double> zahlenwerte;
8
9      for(int i=0; i<5; ++i)
10     {
11         zahlenwerte.push_front(10*i);
12     }
13     cout << "zahlenwerte hat jetzt "
14          << zahlenwerte.size()
15          << " Elemente." << endl;
16     cout << "Vorderstes Element: " << zahlenwerte.front() << endl;
17     cout << "Hinterstes Element: " << zahlenwerte.back() << endl;
18
19     cout << "Loesche alle Elemente von zahlenwerte." << endl;
20     zahlenwerte.clear();
21     cout << "zahlenwerte hat jetzt "
22          << zahlenwerte.size()
23          << " Elemente." << endl;
24     return 0;
25 }
```

- Zeile 2: Auch hier heißt die Include-Datei wie erwartet
- Zeile 5: Eine Liste `zahlenwerte` wird angelegt
- Zeile 11: Mit `push_front()` wird ein Element am Anfang der Liste hinzugefügt. Mit `push_back()` könnte man das am Ende der Liste machen
- Zeile 14: Auch hier gibt der `size()`-Operator die Anzahl der Elemente an
- Zeile 16: Mit `front()` erhält man den Inhalt des vordersten Elements
- Zeile 17: Und mit `front()` erhält man den Inhalt des hintersten Elements
- Zeile 20: Die ganze Liste wird gelöscht mit `clear()`
- Zeile 22: Jetzt sollte `size()` den Wert `0` ergeben (und `empty()` den Wert `true`)

Terminal

```
schueler@debian964:~$ g++ -o liste1 liste1.cpp
schueler@debian964:~$ liste1
zahlenwerte hat jetzt 5 Elemente.
Vorderstes Element: 40
Hinterstes Element: 0
Loesche alle Elemente von zahlenwerte.
zahlenwerte hat jetzt 0 Elemente.
```

10.4.3 Zugriff? Iterator!

Jetzt braucht man noch den Zugriff auf die Elemente zwischen dem vordersten und dem hintersten Element. Dazu gibt es in der STL den so genannten *Iterator*. Es handelt sich um ein Objekt, das zu einem Container zwei Dinge erlaubt:

a) den Zugriff

b) das Wandern von einem Element des Containers zum nächsten

Die STL erlaubt es uns, so einen Iterator **genauso wie eine Zeigervariable** zu benutzen. Wenn `greifer` der Name der Variablen ist, dann:

a) bekommt man mit `*greifer` den Inhalt des Elementes

b) wandert man mit `++greifer` zum nächsten Element

liste2.cpp

```
1  #include <iostream>
2  #include <list>
3  using namespace std;
4
5  int main(void)
6  {
7     list <double> zahlenwerte;
8
9     for(int i=0; i<5; ++i)
10    {
11       zahlenwerte.push_front(5*i+3);
12    }
13
14    list <double> ::iterator greifer;
15
16    greifer=zahlenwerte.begin();
17    while(greifer!=zahlenwerte.end())
18    {
19       cout << *greifer << "; ";
20       ++greifer;
21    }
22    cout << endl;
23    return 0;
24 }
```

- Zeile 14: Der Iterator `greifer` wird vereinbart; der Iterator der Klasse `list` liegt dabei im Namensraum `list::`

- Zeile 16: Mit `begin()` erhält man einen Iterator, der auf das vorderste Element zeigt
- Zeile 17: Aber mit `end()` erhält man einen Iterator, der *hinter* das hinterste Element zeigt; deswegen durchläuft man die Liste, solange man **nicht** bei diesem Element angekommen ist
- Zeile 19: Mit `*greifer` bekommt man den Inhalt dessen, worauf `greifer` zeigt
- Zeile 20: Und hier geht es mit `++greifer` weiter zum nächsten Element

Terminal

```
schueler@debian964:~$ g++ -o liste2 liste2.cpp
schueler@debian964:~$ liste2
23; 18; 13; 8; 3;
```