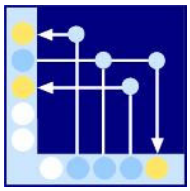
Fakultät Elektronik und Informatik
Studiengang Informatik



Programmieren in C++

Vorlesung im Wintersemester 2017/2018
Prof. Dr. habil. Christian Heinlein

9. Übungsblatt (22. Januar 2018)

Aufgabe 17: Iteratoren

Teilaufgabe 17.a)

Definieren für die Listen aus Aufgabe 12 einen passenden Vorwärtsiteratortyp sowie globale Funktionen `begin` und `end`, um z. B. folgende Verwendungen zu ermöglichen:

```
// Eine Liste mit ganzen Zahlen.
List<int> ls = ......;

// Elemente der Liste ausgeben.
for (int x : ls) cout << x << endl;

// Anzahl der Vorkommen von 5 in der Liste ausgeben.
cout << std::count(begin(ls), end(ls), 5) << endl;
```

✎

```
#include <iterator>
using namespace std;

// Code von Aufgabe 12.
#include "list.cxx"

template <typename T>
struct ListIter : iterator<forward_iterator_tag, T> {
    // Zeiger auf das aktuelle Listenelement.
    Node<T>* ptr;

    // Initialisierung mit Zeiger p bzw. Nullzeiger.
    ListIter (Node<T>* p = nullptr) : ptr(p) {}

    // Aktuelles Listenelement liefern.
    T operator* () const {
        return head(ptr);
    }
```

```
        // Iterator weitersetzen.
        ListIter& operator++ () {
            // Wenn ptr auf das letzte Element einer Liste zeigt,
            // ist es anschließend ein Nullzeiger, was sinnvoll ist.
            ptr = tail(ptr);
            return *this;
        }
        ListIter operator++ (int) {
            ListIter i = *this;
            ++*this;
            return i;
        }


        // Iteratoren vergleichen.
        bool operator== (ListIter that) const {
            return ptr == that.ptr;
        }
        bool operator!= (ListIter that) const {
            return !(*this == that);
        }
    };

    // Iterator liefern, der auf das erste Element der Liste ls verweist.
    // Falls die Liste leer ist, stimmt der Iterator mit dem von end
    // gelieferten Iterator überein.
    template <typename T>
    ListIter<T> begin (List<T> ls) {
        return ListIter<T>(ls);
    }

    // Iterator liefern, der auf das Ende einer beliebigen Liste verweist.
    // (Der Parameter ls wird eigentlich gar nicht benötigt.)
    template <typename T>
    ListIter<T> end (List<T> ls) {
        return ListIter<T>();
    }
```

✎

Teilaufgabe 17.b)

Erstellen Sie eine geeignete Definition von `reverse`, sodass die Anweisung `for (x : reverse(c))` die Elemente der Folge `c` in umgekehrter Reihenfolge durchläuft, sofern der Typ von `c` Elementfunktionen `rbegin` und `rend` besitzt!

Zum Beispiel:

```
vector<int> v = { 1, 2, 3, 4, 5 };
for (int x : reverse(v)) cout << x << endl;
```

Der Inhalt der Folge soll dabei nicht kopiert werden!

✎

```
// Hilfstyp.
// C ist der Typ der zu durchlaufenden Folge
// (der auch const sein kann, wenn die Folge konstant ist).
template <typename C>
struct Reverse {
    // Referenz auf die zu durchlaufende Folge.
    C& c;

    // Initialisierung mit der Folge c.
    Reverse (C& c) : c(c) {}

    // Iteratoren auf den "umgekehrten" Anfang
    // bzw. das "umgekehrte" Ende der Folge c liefern.
    auto begin () { return c.rbegin(); }
    auto end () { return c.rend(); }
};

// Die für den Benutzer sichtbare Funktionsschablone.
template <typename C>
Reverse<C> reverse (C& c) {
    return Reverse<C>(c);
}
```

✎

Aufgabe 18: Initialisiererlisten

Definieren Sie eine Funktionsschablone `cons` analog zu Aufgabe 12, die anstelle eines einzelnen Elements eine Initialisiererliste als ersten Parameter erhält und eine Liste erzeugt, die aus den Elementen dieser Initialisiererliste und der ggf. als zweiten Parameter übergebenen Restliste besteht!

Zum Beispiel:

```
List<int> ls = cons({ 4, 5, 6, });
ls = cons({ 1, 2, 3 }, ls);
```

✎

```
// Code von Aufgabe 17.
#include "listiter.cxx"

template <typename T>
List<T> cons (initializer_list<T> hs, List<T> t = empty<T>()) {
    // Die Initialisiererliste hs rückwärts durchlaufen
    // und Element für Element vor die Liste t setzen.
    for (auto i = hs.end(), b = hs.begin(); i != b; ) {
        t = cons(*--i, t);
    }
    return t;
}
```

✎