Hochschule Aalen
Fakultät Elektronik und Informatik
Studienbereich Informatik

Programmieren in C++

Vorlesung im Sommersemester 2022
Prof. Dr. habil. Christian Heinlein

3. Übungsblatt (4. April 2022)

Aufgabe 4: Weitere Listenfunktionen

Erweitern Sie den Typ `List` aus Aufgabe 3 um folgende Funktionen:

```
// Referenz auf das erste bzw. letzte Element einer konstanten
// Liste liefern.
const string& front () const;
const string& back () const;

// Referenz auf das i-te Element (gezählt ab 0) einer (eventuell
// konstanten) Liste liefern.
// i muss zwischen 0 einschließlich und der Anzahl der Elemente
// ausschließlich liegen; andernfalls ist das Verhalten undefiniert.
string& get (int i);
const string& get (int i) const;

// Ist die Liste momentan leer?
bool empty () const;

// Alle Elemente der Liste entfernen, sodass sie anschließend
// leer ist.
void clear ();

// Element x durch Verschiebung am Anfang bzw. am Ende der Liste
// einfügen.
void push_front (string&& x);
void push_back (string&& x);
```

Da die beiden Varianten von `get` denselben Code ausführen müssen, sollen beide eine gemeinsame Hilfsfunktion (entweder eine Elementfunktion oder eine globale Funktion) verwenden, die diesen gemeinsamen Code enthält. Dies kann durch geeignete An- bzw. Abwesenheit von `const` an den richtigen Stellen der Funktionssignatur erreicht werden.

Die beiden zusätzlichen Varianten von `push_front` bzw. `push_back` sollen das Element `x` ausschließlich durch Verschiebeoperationen zur Liste hinzufügen, d. h. ohne dass es dabei irgendwie kopiert wird. Hierfür muss der Hilfstyp `Node` ebenfalls einen entsprechenden Konstruktor besitzen:

```
Node (string  e) : elem{move(e)} {}
```

Oder geht es eventuell auch ohne diesen zusätzlichen Konstruktor?

Aufgabe 5: Protokollierung von Listenoperationen

Definieren Sie einen Typ `ListLog`, der wie folgt verwendet werden kann:

- Ein Objekt `ll` des Typs `ListLog` wird mit einer Liste `ls` des Typs `List` initialisiert, die bereits beliebig viele Elemente enthalten kann.
- Aufrufe von `push_front`, `push_back`, `pop_front` und `pop_back` auf dem Objekt `ll` werden von diesem an die Liste `ls` weitergeleitet und außerdem in `ll` geeignet protokolliert, damit sie später eventuell wieder rückgängig gemacht werden können.
- Ein Aufruf von `commit` auf dem Objekt `ll` bestätigt die bis jetzt ausgeführten Änderungsoperationen auf der Liste `ls`, indem er alle Einträge des Protokolls verwirft.
- Anschließend könnten erneut Änderungsoperationen auf `ll` ausgeführt werden und wiederum mit `commit` bestätigt werden usw.
- Wenn die Lebensdauer von `ll` endet, werden Änderungsoperationen, die noch nicht mit `commit` bestätigt wurden, durch Ausführung der zugehörigen „inversen“ Operationen auf der Liste `ls` rückgängig gemacht: Aufrufe von `push_front` bzw. `push_back` werden durch Aufrufe von `pop_front` bzw. `pop_back` rückgängig gemacht und umgekehrt.
- Während der Lebensdauer von `ll` müssen alle Änderungsoperationen auf der Liste `ls` über `ll` ausgeführt werden, d. h. weder direkt auf der Liste `ls` noch über andere `ListLog`-Objekte. Konstante Elementfunktionen wie z. B. `front` und `empty` können aber direkt auf `ls` aufgerufen werden.

Zum Beispiel:

```
// Liste ls.
List ls;
......

// Lokaler Block.
{
    // Initialisierung des ListLog-Objekts ll mit der Liste ls.
    ListLog ll{ls};

    // Aufrufe von push_front, push_back, pop_front und pop_back auf ll.
    // Hier kann der Block eventuell vorzeitig durch break, continue,
    // return, goto oder throw verlassen werden.
    ......

    // Bestätigung der zuvor ausgeführten Änderungsoperationen.
    ll.commit();
}

// Normale Weiterverwendung der Liste ls.
......
```

Verwenden Sie zur Protokollierung der Änderungsoperationen ebenfalls eine Liste des Typs `List`! Wenn ein Element der Liste – vor Ausführung einer `pop`-Operation – ins Protokoll geschrieben wird oder umgekehrt – wenn eine `pop`-Operation durch eine entsprechende `push`-Operation rückgängig gemacht wird –, sollen diese Transferoperationen ausschließlich durch Verschiebeoperationen erfolgen.

Objekte des Typs `ListLog` sollen weder kopier- noch verschiebbar sein.