



Die STL nutzen

Datenstrukturen, wie Stapel, Listen und Warteschlange, müssen Sie in C++ nicht mühselig selbst implementieren. Mit der "Standard Template Library" sind solche Strukturen bereits vorgefertigt und einsatzbereit. Also: ehe Sie das Rad neu erfinden, nachschauen was es da gibt!



Mit Containern schneller zum Ziel

OLIVER MÜLLER

Datenstrukturen kommt in der Informatik eine wichtige Rolle zu. Die grundlegende Logik, die hinter derartigen Strukturen steht, ist dabei immer die selbe. Nachdem die Wiederverwendung von Code Einzug in die Software-Entwicklung hielt, dauerte es nicht lange, bis die ersten Library für Datenstrukturen aus dem Boden schossen.

Die modernste Variante sind Container-Klassenbibliotheken. Sie müssen heute keinen Stapel oder Queue selbst implementieren. Durch die standardisierte STL in C++ müssen Sie sich um Datenstrukturen keine Gedanken mehr machen.

Was sind Container?

Ob Sie nun einen Stack für Integers oder Strings benötigen, die Algorithmen zur Verwaltung eines solchen Stapels sind

immer die gleichen. Daher überlegten sich findige Programmierer schon früh Konzepte, wie Listen, Stapel, Deque und Warteschlangen, wiederverwendet werden konnten.

In der strukturierten Programmierung hielten Implementierungen Einzug, die Ihre Daten mit void-Pointern speicherten. Diese void-Pointer mussten dann beim Arbeiten mit den gespeicherten Daten per expliziten Cast (=Typumwandlung) auf den gespeicherten Datentyp "umgebrochen" worden.

Dieses Konzept hatte allerdings seine Tücken. Es war eine Fehlerquelle schlecht in. Wenn Integers gespeichert werden, werden diese über void-Pointer gesichert. Nichts verbietet bei einem solchen Konzept die Rückwandlung des void-Pointer in jeden beliebigen Typ. Einmal in *char** gewandelt und der Absturz war vorprogrammiert.

Mit C++ tauchten die Templates auf.

Templates sind parametrisierbare Klassen. Sie können diesen Klassen bei der Deklaration einen Typ als Parameter übergeben, um so der Klasse mitzuteilen, mit welchen Datentypen Sie intern arbeiten soll. Einer Stapelklasse können Sie so zum Beispiel angeben, dass Sie Strings speichern soll. Durch die Instanziierung dieser Klasse entsteht ein Stack, der nur Strings und wirklich nichts anders speichern kann. Die Fehlerquelle des void-Zeiger ist damit Schnee von gestern.

Diese Klassen, die Datenstrukturen implementieren, nennt man Container. Sie speichern Objekte, ähnlich wie ein Frachtcontainer Waren aufnimmt.

Lange Zeit legte jeder Compiler-Hersteller seine eigene Container-Library bei, die die wichtigen Datenstrukturen, wie Stapel, Listen oder Warteschlangen, bereitstellte. Dieser proprietäre Ansatz hatte erhebliche Probleme. Diese einzelnen proprietären Implementierungen



gen waren zu einander inkompatibel.

Dieses Problem gehört heute auch der Vergangenheit an. Die "Standard Template Library" (STL) – ursprünglich von Hewlett-Packard entwickelt – ist heute ein verbindlicher Standard von C++. Jeder ANSI-konforme C++-Compiler versteht heute die STL.

Containerbau

Eine Übersicht über die Container der STL finden Sie in der Tabelle "Container im Überblick". Sie sehen, dass hier die grundlegenden Datenstrukturen bereits implementiert sind.

In Listing 1 ist ein Beispiel für die Anwendung der STL wiedergegeben. Dieses kleine Programm liest Integer-Werte von der Konsole ein, bis 0 eingegeben wird. Die Integer-Werte werden dabei in einer Liste der Template-Klasse *list* gespeichert, die über mit *int* parametrisiert wurde (siehe Kasten "Wie funktioniert eine Template?").

Zum Speichern verwendet das Programm die Methode *push_front()* des Template-Klasse. Diese Methode speichert den angegebenen Wert am Kopf der Liste. *list* ist eine doppelt verkettete Liste und kann sowohl von Kopf nach

Ende und als auch von Ende nach Kopf arbeiten.

Das Gegenstück zu *push_front()* ist *push_back()*. Diese Methode würde das angegebene Objekt am Ende der Liste speichern.

Dadurch, dass die Werte immer am Anfang der Liste gespeichert werden, werden Sie im zweiten Schritt in umgekehrter Reihenfolge ausgegeben. Die zweite Schleife liest nämlich die Werte aus Liste in der Richtung "Kopf nach Ende" wieder ein. Damit liest es den letzten eingegebenen Wert als erstes, denn dieser liegt bekanntlich am Anfang der Liste.

LISTING 1: INTLIST.CPP

```
#include <iostream>
#include <list>
#include <iterator>

int main()
{
    int num;
    list<int> num_list;

    // Integers einlesen
    cout << "Bitte geben Sie eine Liste von Integers ein (0 = Ende):" << endl;
    cin >> num;
    while(num != 0) {
        num_list.push_front(num);
        cin >> num;
    }

    // Integers per Iterator
    // auslesen
    cout << endl << "Sie haben folgende Integers eingegeben:" << endl;
    list<int>::iterator I = num_list.begin();
    while(I != num_list.end()) {
        cout << (*I) << endl;
        ++I;
    }

    return 0;
}
```

Iteratoren

Zum Zugriff auf die Datenstrukturen verwenden Sie Iteratoren. Über diese können Sie alle Werte aus dem Container nacheinander wieder auslesen.

TIPP Nicht alle Container stellen Iteratoren zur Verfügung. *stack* beispielsweise implementiert keinen, da dies nicht sinnvoll wäre.

Der Iterator ist wieder eine eigene Klasse. Der Name der Klasse ist immer *Container::iterator*. Statt "Container" müssen Sie hier lediglich den Namen des Containers inklusive der Parameter einsetzen.

Im Beispiel wird ein Iterator für den Container *list<int>* über die Zeile

```
list<int>::iterator I = ...
```

deklariert. Diesem Iterator müssen Sie den Anfang der Liste zuweisen. Dies

CONTAINER IM ÜBERBLICK

Container	Header-File	Beschreibung
<code>vector<T></code>	<code><vector></code>	Dynamisches Array zum Speichern von Daten vom Typ T.
<code>list<T></code>	<code><list></code>	Doppelt verkettete Liste mit Daten vom Typ T.
<code>stack<T></code>	<code><stack></code>	Stapel speichert Werte vom Typ T.
<code>queue<T></code>	<code><queue></code>	Schlange für Werte vom Typ T.
<code>deque<T></code>	<code><deque></code>	Doppelendige Schlange für Werte vom Typ T (Deque = double ended queue).
<code>priority_queue<T,D,C></code>	<code><queue></code>	Prioritäten gesteuerte Queue, die Daten vom Typ T speichert und die Daten selbst in der Deque oder dem Vector D speichert. Die Priorität wird über den Operation < des Datentyps T ermittelt. Existiert dieser nicht, muss eine Klasse C angegeben werden, die den Vergleich implementiert.
<code>bitset<T></code>	<code><bitset></code>	Template-Klasse zur Bearbeitung von Bitfolgen fester Größe T.
<code>set<T,C></code>	<code><set></code>	Speichert Mengen vom Typ T. set ist sortiert. Kann über den Operator < nicht sortiert werden, muss über die Klasse C eine Vergleichsklasse angegeben werden, die den Vergleich durchführt. C ist optional.
<code>multiset<T,C></code>	<code><set></code>	Wie set, nur, dass Werte mehrfach (in Bags) gespeichert werden können.
<code>map<K,T></code>	<code><map></code>	Speichert Daten vom Typ T unter dem Schlüssel vom Typ K. (Array mit Index vom Typ K statt einfachem Integer.)
<code>multimap<K,T></code>	<code><map></code>	Wie map, jedoch können unter einem Schlüssel mehrere Daten gespeichert werden.



erfolgt über die Methode `begin()` des Container-Objekts. Auf diese Weise startet der Iterator mit dem ersten Wert in der Liste.

Anschließend liest die `while`-Schleife die einzelne Objekte aus dem Container ein. Das letzte Element ist dann eingelesen, wenn der Iterator `I` den gleichen Wert, wie der Rückgabewert der Methode `end()` des `list`-Objekts angenom-

men hat. Der Zugriff auf das aktuelle Objekt im Container erfolgt über Dereferenzierung des Iterators mit dem Operator `*`. Der Ausdruck `(*I)` in der `while`-Schleife liefert also den aktuell betrachteten Integer-Wert. Damit der Iterator zum nächsten Wert übergeht, ist dieser einfach über den Inkrementoperator `++` hochzuzählen.

UR

```
Terminal
Datei Bearbeiten Einstellungen Hilfe
kernighan:[STL]> ./sample
Bitte geben Sie eine Liste von Integers ein (0 = Ende):
9999
8888
7777
6666
5555
4444
3333
2222
1111
0

Sie haben folgende Integers eingegeben:
1111
2222
3333
4444
5555
6666
7777
8888
9999
kernighan:[STL]> █
```

DAS BEISPIEL IN AKTION, es liest in umgekehrter Reihenfolge zurück.

WIE FUNKTIONIERT EINE TEMPLATE?

Eine Template ist eine Klasse, der über einen Parameter ein Datentyp übergeben wird. Damit ist es möglich eine Schablone für Klassen anzulegen, die über verschiedenen Datentypen die selben Operationen ausführen sollen.

Die Template Klasse `stack` aus der STL ist eine solche parametrisierte Klasse. Sie implementiert einen Stapel, ohne sich auf die darin zu speichernden Datentypen festzulegen. Erst beim Deklarieren von Variablen, legen Sie sich auf den Datentyp fest.

Angenommen der Stapel soll Strings der Klasse `string` speichern, dann deklarieren Sie die entsprechende Variable wie folgt:

```
stack<string> stapel;
```

Die Variable `stapel` enthält nun ein Objekt, das einen Stack darstellt, der Objekte der Klasse `string` speichern kann.

Den Typparameter einer Template-Klasse übergeben Sie also in spitzen Klassen hinter dem Klassennamen bei der Deklaration. Werden mehrere Parameter von der Template-Klasse benötigt, werden diese durch Komma getrennt.

Die Klasse `map` der STL beispielsweise erzeugt eine Art Array, das allerdings nicht zwingend einen Integer-Wert als Index voraussetzt. Sie können zum Beispiel auch Strings als Index angeben. Eine `map`-Instanz, die als Index einen String verwendet und Gleitkommazahlen speichert, würden Sie so deklarieren:

```
map<string,float> feld;
```