Inf Programmieren 1 Übung: STL-Datenstrukturen und Iteratoren

Klaus Kusche

Buchstaben und Worte zählen

Wir bauen unser Programm aus der vorigen Übung so um, dass es exakt dasselbe leistet, aber intern STL-Datenstrukturen verwendet. Zur besseren Strukturierung behalten wir allerdings unsere *bisherige Template-Klasse* **`ValCntArray`** bei (du gehst also am besten von deiner Lösung oder der *Musterlösung* der alten Übung aus) und *verändern nur ihr Innenleben*:

- Wir verwenden das **STL-Template `map`** und brauchen daher den Header **`<map>`**: Eine **`map`** ist eine *Menge von Paaren*. Sie hat den Vorteil, dass sie die Paare automatisch nach dem Schlüsselwert (dem ersten Wert im Paar) *sortiert* liefert und gut suchen und einfügen kann (intern ist eine **`map`** ein Baum von Paaren, sortiert nach dem ersten Wert eines jeden Paares).

  `map` hat *zwei Template-Parameter*:
  Den Typ des ersten und den Typ des zweiten Wertes in jedem Paar.
  In unserem Fall ist der *erste Template-Parameter* (der Typ des ersten Wertes im Paar, d.h. des Suchschlüssels) der *Typ der zu zählenden Daten*, also das **`T`** unseres **`ValCntArray`**-Templates (**`int`**, **`char`** oder **`string`**), und der *zweite Template-Parameter* (der Typ des zweiten Wertes im Paar, d.h. der Daten, die jedem Schlüsselwert zugeordnet werden) ist **`int`**, unser *Zähler*.

  Unsere Template-Klasse braucht genau *eine solche Map als Member-Variable*, alle anderen Member-Variablen (unsere bisherigen Arrays, den Füllstand, und die Hilfstypen dafür) können wir weglöschen.

  Da keine Member mehr zu initialisieren sind (für unsere Map reicht ihr Standard-Konstruktor: Er initialisiert unsere Map auf "leer", d.h. keine Elemente), können wir unseren *Konstruktor weglöschen*: Im Template bleiben nur mehr die beiden öffentlichen Methoden **`count`** und **`print`** übrig.

  Weiters wächst so eine Map nach Bedarf (man muss beim Anlegen keine Größe angeben), wir können also auch den **`size`**-*Parameter unseres Templates entfernen*.

- Unser Hauptprogramm bleibt völlig unverändert (bis auf das Entfernen des **`size`**-Parameters bei den drei **`ValCntArray`**-Variablen).

- Die Schleifenvariable unserer Methode **`print`** stellen wir um auf einen *Iterator* über unsere Map, die Schleife soll *von Anfang bis Ende* unserer Map laufen. Der Iterator zeigt dabei jeweils auf ein *einzelnes Map-Element*, also ein *Paar*, mit **`->first`** erhält man dessen Datenwert, mit **`->second`** den Zählerwert dazu.

  Tipps:

  - Du wirst ein **`typename`** für die Iterator-Deklaration brauchen!
  - **`map`**-Iteratoren kann man nur mit **`==`** und **`!=`** vergleichen, nicht mit **`<`** usw..
  - Wenn du dein **`print`** als **`const`**-Methode deklariert hast, oder wenn du statt **`print`** den **`operator<<`** (mit zweitem Parameter **`const`**) verwendest, dann wirst du einen **`const_iterator`** brauchen!

- Auch die Methode **count** braucht einen solchen *Iterator*, aber nicht für eine Schleife, sondern für das *Ergebnis des* **find**-*Aufrufs* auf unserer Map mit dem zu zählenden Argument-Wert.
    - Wird der Wert *gefunden*, erhöhen wir das **second**-Element (den Zähler) des gefundenen Paares um 1.
    - Wird der Wert *nicht gefunden* (welchen Wert hat der Iterator in diesem Fall?), fügen wir in unsere Map ein neues Element mittels Aufruf von **insert** ein:
      Das neue Element ist ein **pair**, dessen erste Komponente (vom Typ **T**) der *zu zählende Wert* und dessen zweite Komponente (vom Typ **int**) **1** ist.

      Am besten schreibst du einen passenden **pair**-Konstruktor-Aufruf (temporäres Objekt) direkt als Argument-Wert in den **insert**-Aufruf.

Verwende für die Details der verwendeten STL-Klassen **map** und **pair** und die daraus aufgerufenen Methoden **find** und **insert** die Online-Referenz der STL!

Zusatzaufgabe:

- Eigentlich noch einfacher, aber auf den ersten Blick etwas ungewohnt wird das Beispiel, wenn man unserer Klasse **ValCntArray** keine **map** als Member gibt, sondern die Klasse gleich als von **map** (mit den richtigen Typ-Parametern!) *abgeleitete Klasse* definiert:
  Die Klasse hat dann *gar keine Member* mehr, sondern nur mehr unsere beiden Methoden.
  Und überall, wo **map**-Methoden bisher für unser **map**-Member aufgerufen wurden, werden sie jetzt für **this** aufgerufen, denn unser Objekt ist ja selbst die **map**.
  this-> ist in diesem Fall *notwendig*, bei Template-Klassen mit Template-Vaterklassen findet der Compiler die richtigen Methoden der Vaterklasse *nicht automatisch*.

  Damit man von außen nicht direkt mittels **map**-Methoden auf die Daten in **ValCntArray**-Objekten zugreifen kann, sollte man die Klasse **private** von **map** ableiten.
- Wenn du deinen Code noch moderner machen willst, dann kannst du versuchen, **auto** für die Deklaration der *Iteratoren* zu verwenden oder überhaupt die **for(;;)**-*Schleife* mit dem Iterator durch eine **for(:)**-Schleife zu ersetzen.

  Du solltest *beides* probiert haben!
- Weiters lässt sich der *Suchen-und-Erhöhen-oder-Einfügen-Code* noch *kürzer* schreiben, wenn man kein explizites **insert** benutzt, sondern ausnutzt, dass **[]** automatisch ein neues Element anlegt, wenn es nichts findet.

  Aber das **insert** ist eine gute Übung, bitte zuerst mit **insert** versuchen und dann erst auf **[]** umbauen!