Map

Ziel, Inhalt

- Wir lernen die Klasse map aus der Standard-C++ Library kennen und anwenden.

Map

Überblick

Die Klasse map ist eine weitere Container-Klasse aus der Standard-Library. Mit ihr ist es möglich von einem Schlüssel an ein anderes Objekt zu gelangen. Dabei wird das Objekt zusammen mit einem Schlüssel als Pärchen in den Container eingefüllt.

Anwendung

Definition

```
#include <map>
#include <string>
#include <iostream>

class Buch
{
public:
    Buch(const std::string& titel,
         const std::string& autor)
         :_titel(titel), _autor(autor)
    {
    }

    void ausgeben() const
    {
    }

private:
    std::string _titel;
    std::string _autor;
};

// wir definieren einen
// neuen Datentypen, der
// aber einem long enstrpicht
typedef long ISBN;

// Hier verwenden wir eine map
// um mit Hilfe von einer ISBN
// auf ein Buch zu verweisen
typedef std::map<ISBN, Buch> Buchkatalog;
```

Beachte, dass die ISBN Nummer auch eine Klasse sein könnte. Der Einfachheit halber ist es aber nur ein typedef für einen long.
Dieser Datentyp ISBN dient hier als Schlüssel. Wie man Element einfügt und ausliest, sehen wir gleich.

Einfügen von Elementen

```
#include <map>
#include <string>
#include <iostream>

class Buch
{
public:
    // Defaultkonstruktor wird
    // von der map benötigt
    Buch() {}

    Buch(const std::string& titel,
         const std::string& autor)
         :_titel(titel), _autor(autor)
    {
    }

    void ausgeben() const
    {
    }

private:
    std::string _titel;
    std::string _autor;
};

// wir definieren einen
// neuen Datentypen, der
// aber einem long enstrpicht
typedef long ISBN;

// Hier verwenden wir eine map
// um mit Hilfe von einer ISBN
// auf ein Buch zu verweisen
typedef std::map<ISBN, Buch> Buchkatalog;

int main()
{
    ISBN isbn1 = 1233442347;
    ISBN isbn2 = 1734817343;

    Buch einBuch("C++", "Markus");
    Buch nochEinBuch("Mathe", "Fred");

    // Map erzeugen
    Buchkatalog katalog;

    // einfügen in die MAP
    katalog[isbn1] = einBuch;
    katalog[isbn2] = nochEinBuch;

    return 0;
}
```

Die Klasse map überschreibt also den []-Operator (Index-Operator). In den eckigen Klammern steht nun einfach der Datentyp, den man als Key (erstes Template-Argument) definiert hat.

Auslesen von Elementen

Der Index Operator liefert immer eine Referenz auf ein Objekt zurück. Falls das Objekt nicht in der map drin ist, wird es neu erzeugt. Um das zu beweisen definieren wir in der Klasse Buch einen Vergleichsoperator:

```
    // Vergleichsoperator
    bool operator==(const Buch& anderesBuch)
    {
        if(_titel == anderesBuch._titel &&
           _autor == anderesBuch._autor)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

Danach ergänzen wir weiter unten in der main Funktion einen Test mit _ASSERT (oder assert). Hierfür musst du noch <crtdbg.h> (oder <cassert>) mit #include verfügbar machen.

```
    // einfügen in die MAP
    katalog[isbn1] = einBuch;
    katalog[isbn2] = nochEinBuch;
    // Wir holen uns eine
    // Referenz auf das Buch,
    // das bereits in der Kollektion
    // drin ist
    Buch& test = katalog[isbn1];
    // Dieses Buch muss das Buch
    // sein, das wir eingefügt hatten
    _ASSERT(test == einBuch);

    return 0;
}
```

Es ist also sehr einfach auf Elemente zuzugreifen. Ein Problem dabei kann sein, dass wie oben erwähnt der Index Operator immer ein Element zurückliefert. Wenn noch keines mit diesem Schlüssel in der Kollektion war, wird es erzeugt.
Das bedeutet, dass wir so nicht herausfinden können, ob sich ein Element in der Kollektion befindet.

Die find Methode

Um ein Element in der map zu finden, bietet diese die Methode find an. Die find Methode gibt als Rückgabewert einen iterator zurück, was einem Zeiger auf ein Schlüssel/Element Pärchen entspricht. Was das für ein Pärchen ist, sehen wir im nächsten Abschnitt. Wird das Element nicht gefunden, hat der zurückgegebene iterator den gleichen Wert wie der iterator, den die Methode end() zurückgibt, er zeigt also „hinter“ die Kollektion. Hier das Beispiel:

```
ISBN isbn3 = 999888777;
if(katalog.find(isbn3) == katalog.end())
{
    // Buch mit der isbn3 wurde
    // nicht gefunden
}
else
{
    // Buch gefunden
}

if(katalog.find(isbn2) != katalog.end())
{
    // Buch mit isbn2 wurde
    // gefunden
}
```

Die Elemente in der map

der value_type in der map

In der map wird immer ein Pärchen aus Schlüssel und Element abgelegt. Ein solches Pärchen ist in unserem Beispiel so definiert:

```
typedef std::pair<ISBN, Buch> BuchkatalogElement;
```

Es ist nicht nötig den Typen BuchkatalogElement selber zu definieren, dieser wird automatisch definiert, wenn wir die map verwenden und heisst value_type.

```
Buch drittesBuch("Roman", "Sepp");
Buchkatalog::value_type element(isbn3, drittesBuch);
```

Der value_type besteht aus zwei Elementen, die ganz einfach first und second heissen. In der map ist der first der Schlüssel und der second das Element. Siehe hierzu auch „Die iterator Typen“ weiter unten.

Einfügen mit insert

Ein solches value_type Pärchen lässt sich auch mit insert in die map einfügen:

```
Buch drittesBuch("Roman", "Sepp");
```

```
Buchkatalog::value_type element(isbn3, drittesBuch);

katalog.insert(element);
```

Die iterator Typen

Wie bei den anderen Kollektionen, gibt es auch bei der map iteratoren. Diese iteratoren zeigen in der map auf den value_type, also immer auf ein Pärchen. Um alle Bücher in unserem Katalog auszugeben verwenden wir folgende Schleife:

```
Buchkatalog::iterator it = katalog.begin();
Buchkatalog::iterator end = katalog.end();

for( ; it != end; ++it)
{
    // der iterator ist ein
    // Zeiger auf ein Buchkatalog::value_type
    Buchkatalog::value_type paar = *it;
    ISBN isbn = paar.first;
    Buch buch = paar.second;
    buch.ausgeben();
}
```

Das ganze lässt sich auch kürzer schreiben:

```
Buchkatalog::iterator it = katalog.begin();
Buchkatalog::iterator end = katalog.end();

for( ; it != end; ++it)
{
    // der iterator ist ein
    // Zeiger auf ein Buchkatalog::value_type
    it->second.ausgeben();
}
```

Gefundenes Element

Die Methode find gibt wie bereits gesehen einen iterator zurück.

```
Buchkatalog::iterator gefunden = katalog.find(isbn3);
if(gefunden != katalog.end())
{
    // gefundenes Element ausgeben
    gefunden->second.ausgeben();
}
```

Übung Medienshop erweitern

Ergänze den Medienshop um die Möglichkeit mit dem Namen nach einem Medium zu suchen. Verwende ab jetzt einen map um Medien zu speichern!