Übung CD-Liste

Einführung

Ich hoffe durch dieses Beispiel einige Themen aufzugreifen, die wir in den letzten Lektionen kennengelernt haben und diese in einem konkreten Beispiel anzuwenden. Es geht hier darum ein kleines CD-Programm zu schreiben, das zwar nur ganz wenige Möglichkeiten bietet und darum hoffentlich auch übersichtlich bleibt.

Erster Schritt: Klasse CD

Die Klasse CD soll also die Daten zu einer CD enthalten. Für unsere Zwecke genügt der Name des Interpreten, der Titel der CD und deren Länge in Minuten und Sekunden. Da wir die CD sozusagen in einer Datenbank aufbewahren wollen soll jede CD noch eine eindeutige Nummer haben, eine ID. Die Klasse sieht also ziemlich ähnlich aus, wie wir das bereits kennen. Hier ein Vorschlag:

```
struct Laenge
{
    long Minuten;
    long Sekunden;
};


class CD
{
    public:
        // default Konstruktor
        CD();
        // Destruktor
        ~CD();

        void setzeInterpret(const string& interpret);
        void setzeTitel(const string& Titel);
        void setzeLaenge(const Laenge& Laenge);

    private:
        static long s_nextId;

        long    m_id;
        string  m_Interpret;
        string  m_Titel;
        Laenge  m_Laenge;
};
```

Die statische Variable s_nextId wird beim Erzeugen von einer CD um eins erhöht. Sie wird in der .cpp Datei mit 0 initialisiert.

Zweiter Schritt: Klasse CDListe

Wir wollen zur Verwaltung von CD's gleich eine Klasse CDListe schreiben. Bei dieser Liste soll es möglich sein eine CD hinzuzufügen. Es soll auch möglich sein nach einer CD zu suchen. Schliesslich soll es möglich sein die ganze Liste auszugeben. Die CDListe merkt sich die CD's in einem Array von CD's. Eine Klasse nach diesen Vorgaben könnte etwas so aussehen:

```
const long MaxCDs = 50;


class CDListe
{
    public:
        CDListe();
        ~CDListe();

        void ausgeben();
        bool hinzufuegen(const CD& cd);
        CD& suchen(const string& suchBegriff);

    private:
        CD m_CDs[MaxCDs];

        long m_Anzahl;
};
```

Die Konstante MaxCDs bestimmt die maximale Anzahl CD's in unserer CDListe. Wenn wir ein Objekt der Klasse CDListe erzeugen, wird ein Array von MaxCDs CDs erzeugt. Diese CD's sind aber noch „undefiniert", das heisst sie enthalten noch keine wirklichen Daten. Um eine CD hinzuzufügen wird zuerst eine CD erzeugt (zum Beispiel in der main-Funktion). Danach kann diese man diese CD der Funktion hinzufuegen mitgeben. Etwa so:

```
#include "CD.h"
#include "CDListe.h"

int main()
{
    CDListe liste;

    CD eineCD;
    eineCD.setzeInterpret("Pearl Jam");
    eineCD.setzeTitel("vs.");
    Laenge laenge;
    laenge.Minuten = 45;
    laenge.Sekunden = 23;
    eineCD.setzeLaenge(laenge);

    liste.hinzufuegen(eineCD);

    return 0;
}
```

Die Liste bestimmt aufgrund ihrer Variable m_Anzahl an welcher Stelle im Array die neue CD hinkommmt. Falls es noch genügend Platz hat für die

neue CD gibt die Funktion *true* zurück sonst *false*. Der Code für diese Funktion sieht ungefähr so aus:

```
bool CDListe::hinzufuegen(const CD& cd)
{
    bool resultat = false;

    if(m_Anzahl < MaxCDs)
    {
        m_CDs[m_Anzahl] = cd;
        m_Anzahl++;
        resultat = true;
    }

    return resultat;
}
```

Die Funktion suchen ist ein wenig aufwendiger, wir müssen dafür die Klasse CD noch anpassen. Das beste wäre es wir könnten eine CD fragen ob ein Suchbegriff passt oder nicht. Dann könnten wir die Funktion *suchen* ziemlich einfach implementieren.

```
CD& CDListe::suchen(const string& suchBegriff)
{
    static CD leereCD;  // diese wird nur einmal
                        // erzeugt !
    leereCD.setzeInterpret("LEER");

    for(long index = 0; index < m_Anzahl; ++index)
    {
        if(m_CDs[index].passt(suchBegriff))
        {
            // falls eine CD "passt"
            // springen wir aus der Funktion
            // und geben diese CD als
            // "return"-Wert
            return m_CDs[index];
        }
    }

    // Falls wir bis hierher kommen wurde keine
    // passende CD gefunden. Wir geben stattdessen
    // diese statische "leere" CD zurück
    return leereCD;
}
```

Dafür müssen wir in der Klasse CD noch diese Funktion *passt* implementieren. Diese Funktion könnte einfach überprüfen, ob der Begriff gleich dem Interpreten oder dem Titel ist.
Die Funktion *ausgeben* ist auch ziemlich einfach zu lösen. Wir überlassen die Ausgabe auf dem Bildschirm der Klasse CD gleich selber. Wir schreiben in der CDListe einfach eine Schleife, die für jede CD *ausgeben* aufruft. Es gilt also in der Klasse CD auch eine Methode *ausgeben* zu definieren.

Dritter Schritt: Sortieren

Es soll möglich sein unsere CDListe nach ID, nach Titel, Laenge oder Interpret zu sortieren. Als Sortieralgorithmus verwenden wir den Bubble-Sort Algorithmus, den wir bereits kennen. Dieser Algorithmus vergleicht zwei Elemente und tauscht diese so aus, dass das kleinere „links“ ist. Auch hier soll eine CD selber bestimmen, ob sie kleiner ist als eine andere. Wir fügen in die Klasse CD eine Methode vergleiche ein. Zusätzlich geben wir auch an, was von den CD’s verglichen werden soll. Hierfür definieren wir einen *enum,* eine Aufzählung, die angibt welches Element verglichen werden soll. Etwa so:

```
class CD
{
    public:
        // etc.
        enum Kriterium
        {
            Id,
            Interpret,
            Titel,
            Laenge
        };

        bool kleinerAls(const CD& andereCD, Kriterium was);
        // etc.
```

Definiere auch eine Funktion *SwapCDs,* die zwei CD’s vertauscht. In einer Schleife in der CDListe – *sortieren* Funktion werden zwei CD’s genommem und miteinander verglichen und bei Bedarf getauscht.

Schreibe eine nette main-Funktion, die dem Benutzer die Möglichkeit gibt CD’s einzugeben, zu sortieren und anzuzeigen! Das funktioniert am besten mit einem Menu. Du kannst sogar eine Klasse Menu schreiben!