Abend 22
Operatoren überladen

Ziel, Inhalt

- Wir schreiben eine eigene String Klasse, deren Objekte wir mit dem +-operator zusammenfügen können oder die mit dem == operator verglichen werden können. Für die Ausgabe soll ein Objekt unserer String-Klasse mit dem Einfüge-operator (<<) ausgegeben werden können.

Die Klasse MyString

Diese Klasse ist ähnlich aufgebaut wie die Klasse Notenliste oder die Klasse CDListe, die wir bereits kennen. Wir werden in unserer MyString-Klasse ein Array von char verwalten.

Übung Klasse MyString

Erzeuge eine Header-Datei für diese Klasse MyString und definiere die Klasse analog zu den Listenklassen, die wir bereits kennen. Definiere eine Konstante, die die maximale Anzahl Zeichen in einem unserer MyString-Objekte setzt. Wähle ruhig einen recht grossen Wert (100 ist z.B. OK).
Die Klasse soll einen Default-Konstruktor haben. Zusätzlich soll die Klasse einen Kosntruktor mit Parameter haben. Der Parameter soll ein C-String sein, das heisst am besten ein *const char**.
Als Datenelemente ist ein char-Array vorhanden.
Braucht es aber für die Länge des Strings auch ein Datenelement, das angibt wieviele Zeichen der String braucht? Oder gibt es einen anderen Weg die Länge unseres Strings zu bestimmen?
Definiere eine Methode *ausgeben(ostream& out)* und eine Methode *holeLaenge*, die angibt wie lange der String ist. Sind diese Methoden konstant (ändert sich ein Datenelement durch diese Methoden)?
Das Array von char's soll im Konstruktor und im Konstruktor mit Parameter zuerst mit 0 gefüllt werden. Schreibe dafür eine kleine Methode mit einer Schleife (oder verwende *memset*). Diese Methode gehört vorerst in den privaten Teil der Klasse.
Erzeuge nun auch die .cpp-Datei für diese Klasse und implementiere die Methoden.
Nützlich dabei sind einige Methoden aus der C-Library (#include „string.h"), wie zum Beispiel

```
// Funktionsprototyp
char* strncpy(char* ziel, const char* quelle, int anzahl);
```

Diese kopiert maximal *anzahl* Zeichen vom C-String quelle in unser Array. Diese Funktion ist nützlich um den Konstruktor mit Parameter zu implementieren.
Definiere und implementiere auch einen Kopierkonstruktor.

C-Strings zusammenfügen

In der letzten Übung galt es eine eigene Funktion strcat zu schreiben. Mit dieser ist man in der Lage zwei C-Strings zusammenzufügen, sofern der C-String genug Platz bietet. Hier ein Beispiel:

```
#include <string.h> // beachte das .h!
#include <iostream>

using namespace std;

int main()
{
    char string1[50] = "Hallo ";
    const char* string2 = "Welt";

    strcat(string1, string2);

    cout << string1 << endl;

    return 0;
}
```

Beachte, dass wir hier die Datei „string.h“ includieren. Diese stammt noch aus alten C-Zeiten und enthält die Prototypen für Funktionen wie strcat, strlen, etc. Beachte auch, dass der *string1* genügend Speicher reserviert, so dass der zusammengesetzte C-String darin Platz hat (andernfalls hätten wir ein Problem).
Unsere Klasse MyString soll auch diese Möglichkeit bieten, etwa in dieser Art:

```
#include "MyString.h"
#include <iostream>

using namespace std;

int main()
{
    MyString string1("Hallo ");
    MyString string2("Welt");

    string1.hinzufuegen(string2);

    string1.ausgeben(cout);

    return 0;
}
```

Übung Methode *hinzufuegen(const MyString& anderer)*

Schreibe jetzt also eine Methode *hinzufuegen* und verwende hierfür die vordefinierte Funktion

```
// Funktionsprototyp
char* strncat(char* ziel, const char* quelle, int anzahl);
```

Diese Funktion fügt maximal anzahl Zeichen aus quelle an unser ziel an.

Operatoren überladen

Wenn wir einen Operator überladen geben wir einem Operator (zum Beispiel der operator +) eine neue Bedeutung. Wir können dadurch einem operator für eigene Datentypen (eigene Klassen) eine Bedeutung geben. Ein gutes Beispiel ist die Klasse MyString. Um zwei MyString-Objekte zusammenzufügen wäre anstelle der Schreibweise oben (mit der Methode *hinzufuegen*) folgende Schreibweise möglich:

```
#include "MyString.h"
#include <iostream>

using namespace std;

int main()
{
    MyString string1("Hallo ");
    MyString string2("Welt");

    string1 += string2; // operator +=

    cout << string1; // operator <<

    return 0;
}
```

Hierbei ist es wichtig festzustellen welche Datentypen jeweils links und rechts vom operator stehen. In unserem Beispiel ist beim operator+= links eine Objekt der Klasse *MyString* und rechts ebenso. Beim zweiten ist links eine Objekt der Klasse *ostream*. Diesen operator werden wir vorerst noch nicht definieren.
Ein operator für die Klasse MyString zu überladen funktioniert so:

```
class MyString
{
    public:
    ...
    void operator+=(const MyString& anderer);
    ...
};

void MyString::operator+=(const MyString& anderer)
{
    // vorhandene Methode aufrufen
    hinzufuegen(anderer);
}
```

So gesehen unterscheiden sich Operatoren nicht von beliebigen anderen Funktionen und Methoden. Im allgemeinen erwarten wir aber von einem operator ein bestimmtes Verhalten (+ sollte so etwas wie eine Addition durchführen).

Übung Operatoren überladen

Überlade jetzt folgende Operatoren für die Klasse MyString:

- operator +=
- operator +
- operator ==

Die Operatoren sehen in der Klassendefinition etwa so aus:

```
class MyString
{
    public:
    ...
    void operator+=(const MyString& anderer);
    bool operator==(const MyString& anderer);
    MyString operator+(const MyString& anderer);
    ...
};
```

Der Operator + gibt ein MyString-Objekt zurück damit folgender Code geschrieben werden kann:

```
#include "MyString.h"
#include <iostream>

using namespace std;

int main()
{
    MyString string1("Hallo ");
    MyString string2("Welt");
    MyString string3;

    string3 = string1 + string2; // operator +

    bool b = (string1 == string2) // Vergleich

    return 0;
}
```

string1 + string2 erzeugt einen neuen string, der dann unserem string3-Objekt zugewiesen wird. Der Vergleichsoperator (operator ==) gibt einen bool zurück, so wie wir das erwarten würden.

Weitere Operatoren überladen

Der Operator << kann auf diese Art nicht überladen werden, denn links steht im allgemeinen ein Objekt einer Klasse, wie ostream. Diese Klasse können wir nicht mehr ändern, wir haben den Source-Code nicht zur Verfügung. Es gibt aber auch die Möglichkeit Operatoren global also unabhängig von einer Klasse zu überschreiben. Der Compiler sucht nämlich irgend eine Funktion, die mit den Datentypen zurechtkommt. Bei einer Zeile wie dieser:

```
cout << string1; // operator <<
```

sucht der Compiler eine Methode in der Klasse ostream (cout ist ein Objekt dieser Klasse), die als Argument ein Objekt der Klasse MyString nimmt, findet diese nicht. Wenn wir dem Compiler eine solche Funktion anbieten nimmt er diese:

```
// Funktionsprototyp
void operator<<(ostream& out, const MyString& string);
```

Diese Funktion passt, denn als linkes Argument haben wir einen ostream und rechts ein MyString-Objekt. Diese Funktion gehört aber nicht zur Klasse MyString, trotzdem wäre es nützlich wir könnten sie so implementieren:

```
void operator<<(ostream& out, const MyString& string);
{
    out << string.m_string;
}
```

Das würde wunderbar funktionieren, denn das Datenelement *m_string* von *MyString* ist vom Typ char[], ein Datentyp für den man den <<-Operator in der Klasse *ostream* findet.
Ein kleines Problem bleibt. Da *m_string* aber ein privates Datenelement ist, darf diese allgemeine Funktion nicht darauf zugreifen. Das Datenelement aus diesem Grund *public* zu machen, wäre aber eine EXTREM BÖSE SACHE! Es gibt für genau dies eine nette kleine Lösung in C++

friend

Es ist möglich einer Funktion oder einer anderen Klasse den Zugriff auf private Datenelemente zu geben, indem man sie als *friend* deklariert. Das sieht dann für diesen operator<< so aus:

```
class MyString
{
    public:
    ...
    void operator+=(const MyString& anderer);
    bool operator==(const MyString& anderer);
    MyString operator+(const MyString& anderer);

    friend void operator<<(ostream& out,
                           const MyString& string);
    ...
};
```

Übung operator<<(ostream, MyString)

Überlade jetzt also den Operator << so dass es möglich ist folgenden Code zu schreiben:

```
#include "MyString.h"
#include <iostream>

using namespace std;

int main()
{
    MyString string1("Hallo ");
    MyString string2("Welt");

    string1 += string2; // operator +=

    cout << string1; // operator <<

    return 0;
}
```

Zusätzliche Übung

Erweitere die Lösung zur Übung Notenliste so dass es den Operator<< für die Klasse Notenliste und die Klasse Note gibt. Anstelle von *ausgeben* ist es dann möglich zu schreiben:

```
int main()
{
    cout << "Fach ? ";
    string Fach;
    cin >> Fach;

    Notenliste eineNotenliste(Fach);

    /////// SNIP ///////

    // Hier in der Konsole ausgeben
    cout << eineNotenliste;

    getch();

    return 0;
}
```