Vektoren

Ziele

Wir lernen heute auf einfache Art viele Objekte eines Datentypen hintereinander im Speicher anzulegen.

Definition von Vektoren

Mit der Vektor-Schreibweise können wir in nur einer Zeile viele Objekte von einem Datentypen erzeugen. Als Beispiel nehmen wir eine Messreihe, bei der tausend Messwerte erfasst und gespeichert werden müssen. Die Messwerte sind als double zu speichern. Natürlich könnten wir hingehen und tausend double Variablen definieren :

```
double wert1 = 0.0;
double wert2 = 0.0;
double wert3 = 0.0;
...
```

Das macht aber keinen Sinn, viel einfacher ist folgendes :

```
double werte[1000]; // Vektor für tausend double werte
```

Zugriff auf Vektorelemente

Um auf die einzelnen Werte zuzugreifen verwenden wir folgende Schreibweise :

```
werte[0] = 0.0;
werte[1] = 1.0;
...
werte[999] = 999.0;
```

Die Zahl in den [] eckigen Klammern ist der sogenannte Index. Als Index können Ganzzahlen verwendet werden, sowie Ganzzahlvariablen. Genauso können aber auch Aufzählungen (enum) verwendet werden. Zu beachten ist folgendes: Der erste Wert im Vektor hat den Index 0! Der letzte Wert im Vektor hat als Wert die Grösse des Vektors minus 1!

Beispiel 1

```
// fuer rand, Zufallszahlen
#include <stdlib.h>
// fuer time, die Zeit
#include <time.h>

#include <iostream>

using namespace std;

int main()
{
     // Die Funktion time liefert
     // die Anzahl Sekunden seit dem
     // 1. Januar 1970
     long sekunden = time(0);
     // Mit srand wird der Zufallszahlen-
     // generator "gefüttert"
     srand(sekunden);

     // Vektor von tausend Werten
     // vom Datentyp long
     long Zufallszahlen[1000];

     for(long index = 0; index < 1000; ++index)
     {
          Zufallszahlen[index] = rand();
     }
     // Die Werte sind jetzt im Vektor
     // Wir geben sie wieder aus :
     for(index = 0; index < 1000; ++index)
     {
          cout << Zufallszahlen[index] << endl;
     }

     return 0;
}
```

Beispiel 2

```
#include <string>
#include <iostream>
#include <conio.h>

using namespace std;

enum WochentagCodes
{
     Montag = 0,
```

```
     Dienstag,
     Mittwoch,
     Donnerstag,
     Freitag,
     Samstag,
     Sonntag
};


void initDeutsch(string Tage[7])
{
     Tage[Montag] = "Montag";
     Tage[Dienstag] = "Dienstag";
     Tage[Mittwoch] = "Mittwoch";
     Tage[Donnerstag] = "Donnerstag";
     Tage[Freitag] = "Freitag";
     Tage[Samstag] = "Samstag";
     Tage[Sonntag] = "Sonntag";
}

void initItalienisch(string Tage[7])
{
     Tage[Montag] = "Lunedi";
     Tage[Dienstag] = "Martedi";
     Tage[Mittwoch] = "Mercoledi";
     Tage[Donnerstag] = "Giovedi";
     Tage[Freitag] = "Venerdi";
     Tage[Samstag] = "Sabato";
     Tage[Sonntag] = "Domenica";
}

int main()
{
     string Wochentage[7];

     cout << "Hallo, ich kann Deutsch ";
     cout << "und Italienisch !" << endl;

     cout << "[1] Fuer Deutsch" << endl;
     cout << "[2] Fuer Italienisch" << endl;

     int wahl = getch();

     if('2' == wahl)
     {
          initItalienisch(Wochentage);
     }
     else
     {
          initDeutsch(Wochentage);
     }
```

```
     for(long index = 0; index < 7; ++index)
     {
          cout << Wochentage[index] << endl;
     }

     cout << endl << "    Taste";

     getch();

     return 0;
}
```

Initialisierung von Vektoren

Es ist möglich Vektoren direkt zu initialisieren:

```
float zahlen[4] = {1.2, 2.3, 3.4, 4.5 };
```

Die einzelnen Werte sind mit Kommas voneinander getrennt. Mit einer Initialisierungsliste ist es auch nicht nötig die Länge des Vektors anzugeben:

```
int vektor[] = { 3, 5, 7, 6, 8};    // Vektor mit 5 int-Werten
```

Wird die Anzahl angegeben, enthält aber die Initialisierungsliste weniger Elemente als der Vektor, werden die restlichen mit 0 belegt. Auf diese Art ist es einfach ein Vektor mit 0 zu initialisieren:

```
int vieleZahlen[10000] = {0};// alle mit Null initialisieren
```

Ein Vektor kann nicht einem anderen Vektor zugewiesen werden. Es ist also nicht möglich solchen Code zu schreiben:

```
int zahlen[] = {3, 5, 7};
int auchZahlen[3] = zahlen; // geht nicht !!!! FALSCH
```

Um diesen Vektor zu kopieren müsste man folgenden Code schreiben, (man spricht von elementweisem Kopieren):

```
int zahlen[] = {3, 5, 7};
int auchZahlen[3];
for(long index = 0; index < 3; ++index)
{
    auchZahlen[index] = zahlen[index];
}
```

C-Strings

C-Strings sind Vektoren von char-Elementen mit einer speziellen Eigenschaft, das letzte Element enthält den Wert 0. Das 0 bezeichnet das Ende eines C-Strings. Einem solchen C-String kann mit Text gefüllt werden. zum Beispiel so:

```
char einText[] =“Hallo Welt“; // C-String
```

Hinter dem t von “Welt” setzt der Compiler eine 0 ein. Dadurch braucht der Text 11 Bytes, obwohl der Text nur 10 Zeichen lang ist. Nur so ist es möglich das Ende eines Textes zu markieren. Einen solchen Text kann man auch mit cout ausgeben:

```
#include <iostream>

using namespace std;

int main()
{
    char einCString[] = "Hallo Welt";

    cout << einCString << endl;

    return 0;
}
```

Das Programm weiss nur aufgrund der abschliessenden 0 im Speicher hinter dem ‚t’ wo die Ausgabe beendet werden muss. Anders ist es nicht möglich bei einem C-String die Länge festzustellen. Wichtig ist auch, dass man immer daran denkt, dass am Ende diese 0 hinzugehört und man auch genug Speicher anlegt:

```
char text[5] = „Hallo“; // FALSCH, mit der 0 braucht es 6
                        // char’s
```

Von der C-Sprache her gibt es einige Funktionen, die mit den C-Strings arbeiten (die string-Klasse gibt es erst seit C++!), diese beginnen meist mit „str“ (z.B. strlen(), strcpy(), strcmp()).

Klassen-Arrays

Erzeugt man einen Vektor mit einer Klasse als Datentyp, spricht man von einem Klassen-Array. Der Vektor von strings aus dem Beispiel oben ist also ein sogenanntes string-Array. Man kann solche Arrays auch initialisieren:

```
string Wochentage[] =
{
    string(„Montag“),
    string(„Dienstag“),
    string(„Mittwoch“),
...
};
```

Beispiel mit unserer ZeichenFlaeche

Auch mit den Klassen, die wir aus der Zeichnen-Übung kennen ist es möglich Vektoren (Arrays) zu bilden:

```
ZeichenFlaeche flaechen[] =
{
    ZeichenFlaeche(100,200),
    ZeichenFlaeche(200,300),
    ZeichenFlaeche(300,400)
};
```