Kapitel 10

Felder / Arrays

Felder

- Bisher wurden nur Variablen verwendet, die einen einzelnen Wert aufnehmen können.
- Diese Variablen werden auch Skalare genannt.
- Aus einer Menge von Skalaren gleichen Datentyps kann man ein Feld (array) bilden.
- Die einzelnen Skalare werden dann durch Indizes unterschieden.

Felder

- In der Mathematik werden Felder z.B. für die Darstellung von Vektoren und Matrizen verwendet.

$$v = (v_1, v_2, ..., v_n)$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- Vektoren sind eindimensionale und Matrizen zweidimensionale Felder.

Definition von Feldern

- Jedes Feldelement hat den gleichen Datentyp. Er steht vor dem Feldnamen.
- Hinter dem Feldnamen steht die Angabe zur Dimensionierung.
- Für jede Dimension wird die Dimensionierung in ein Paar eckige Klammern gesetzt.
- Sie gibt die Anzahl der Feldelemente an, die abgespeichert werden können.
- Es handelt sich um einen ganzzahligen konstanten Ausdruck.
- Der Index einer Dimensionierung beginnt immer mit 0;

Eindimensionale Felder

- In folgendem Beispiel wird ein eindimensionales Feld mit dem Namen *zahlen* definiert.
- Das Feld hat 20 Elemente.
- Alle Elemente sind vom Datentyp *int*.
- Die Indizes liegen im Bereich von 0 bis 19.

```
int zahlen[20];
```

Eindimensionale Felder

- In folgendem Beispiel wird ein eindimensionales Feld mit dem Namen *werte* definiert.
- Das Feld hat 10 Elemente.
- Alle Elemente sind vom Datentyp *double*.
- Die Indizes liegen im Bereich von 0 bis 9.

```
const int MAXINDEX = 9;
double werte[MAXINDEX+1];
```

Eindimensionale Felder

- Der Zugriff auf die einzelnen Feldelemente erfolgt unter Verwendung des Indexoperators [].
- Der Index muss stets innerhalb des definierten Wertebereiches liegen.
- Wird der Wertebereich verlassen, können schwerwiegende Fehler auftreten.
- Die Zuweisung eines Feldes an ein anderes ist nicht möglich.

```
zahlen[0], zahlen[1], …, zahlen[19]
werte[0], werte[1], …, werte[19]
```

Eindimensionale Felder

- In folgendem Beispiel wird ein eindimensionales Feld mit 10 Elementen definiert.
- Die einzelnen Elemente werden mit den Werten 0, 1, …, 9 beschrieben.

```
const int ANZAHL = 10;
double feld[ANZAHL];
for (int i=0; i < ANZAHL; i++)
    feld[i] = i;
```

Initialisierung von eindimensionalen Feldern

- Felder können bei deren Definition initialisiert (also mit einem Startwert belegt) werden.
- Hierzu wird ein Wert für jedes einzelne Feldelement angegeben.

```
double feld[5] = {0, 2, 4, 6, 8};
```

Initialisierung von eindimensionalen Feldern

- Wird die Initialisierung am Anfang nicht durchgeführt, kann auf die einzelnen Feldelemente erst zugegriffen werden, sobald diese mit Werten gefüllt wurden.
- Ein

```
double feld[5];
cout << feld[0];
```

führt zum Absturz des Programms, da in dem feld[0] noch kein Wert gespeichert wurde.

Initialisierung von eindimensionalen Feldern

- Es ist immer ratsam, ein Feld zu initialisieren.
- Eine Teilinitialisierung ist möglich. Die restlichen, nicht angegebenen Felder, werden mit 0 initialisiert:

```
double feld[5] = {0, 2, 4};
```

- Leere geschweifte Klammern bewirken eine Initialisierung aller Feldelemente mit 0:

```
double feld[5] = {};
```

Initialisierung von eindimensionalen Feldern

- Ein

```
double feld[5] = {};
cout << feld[0];
```

funktioniert demnach und liefert als Ausgabe:

```
0.0
```

Kopieren von eindimensionalen Feldern

- Die Zuweisung eines Feldes an ein anderes ist nicht möglich.

```
double feldA[5] = {};
double feldB[5] = {1, 2, 3, 4, 5};

feldA = feldB                    // Fehler !!!
```

Kopieren von eindimensionalen Feldern

- Man muss die einzelnen Elemente nacheinander kopieren.

```
double feldA[5] = {};
double feldB[5] = {1, 2, 3, 4, 5};

for (int i=0; i<5; i++)
    feldA[i] = feldB[i];
```

Operator *sizeof*

- Der Operator *sizeof* ermittelt die Größe des übergebenen Datentyps oder des übergebenen Objektes in Bytes.
- *sizeof(feld)* liefert 40. Das Feld im vorherigen Beispiel hat 5 Elemente mit jeweils 8 Bytes.
- *sizeof(int)* liefert 4.

Eindimensionales Feld - Beispiel

- In folgendem Beispiel wird der Benutzer zur Eingabe von maximal 10 Zahlen aufgefordert.
- Die Eingabe wird mit der Eingabe eines Buchstabens beendet.
- Danach werden die Zahlen sortiert und ausgegeben.
- Siehe Beispiel 10 – Programm zu eindimensionalen Feldern (auf der Webseite zu finden).

Mehrdimensionale Felder

- Felder können mit bis zu 256 Dimensionen definiert werden.
- Die am häufigsten verwendeten mehrdimensionalen Felder sind zweidimensionale Felder.
- Für jede Dimension muss die Anzahl der Elemente angegeben werden.

```
int matrix[3][3];
```

Initialisierung von mehrdimensionalen Feldern

- Es ist auch möglich, mehrdimensionale Felder zu initialisieren.

```
int matrix[3][3] = {0, 2, 4, 1, 3, 5, 7, 8, 9};
```

Bedeutung:

$$matrix = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \\ 7 & 8 & 9 \end{bmatrix}$$

Initialisierung von mehrdimensionalen Feldern

- Durch zusätzlich gesetzte Klammern kann die Zuordnungsreihenfolge verdeutlicht oder geändert werden.

```
int matrix[3][3] = {{0, 2, 4}, {5}, {7, 8, 9}};
```

Bedeutung:

$$matrix = \begin{bmatrix} 0 & 2 & 4 \\ 5 & 0 & 0 \\ 7 & 8 & 9 \end{bmatrix}$$

Kopieren von mehrdimensionalen Feldern

- Das Kopieren von mehrdimensionalen Feldern funktioniert analog zu der Methode bei eindimensionalen Feldern.

```
double matrixA[3][3] = {};
double matrixB[3][3] = {1, 2, 3, 4};

for (int i=0; i<3; i++)
    for (int j=0; j<3; j++)
        matrixA[i][j] = matrixB[i][j];
```

Mehrdimensionale Felder - Beispiel

- In folgendem Beispiel werden eine 3x3-Matrix und ein Vektor mit 3 Elementen definiert.
- Es wird der Vektor in die zweite Zeile der Matrix kopiert und die Matrix dann auf den Bildschirm ausgegeben.
- Siehe Beispiel 11 – Programm zu mehrdimensionalen Feldern (auf der Webseite zu finden).