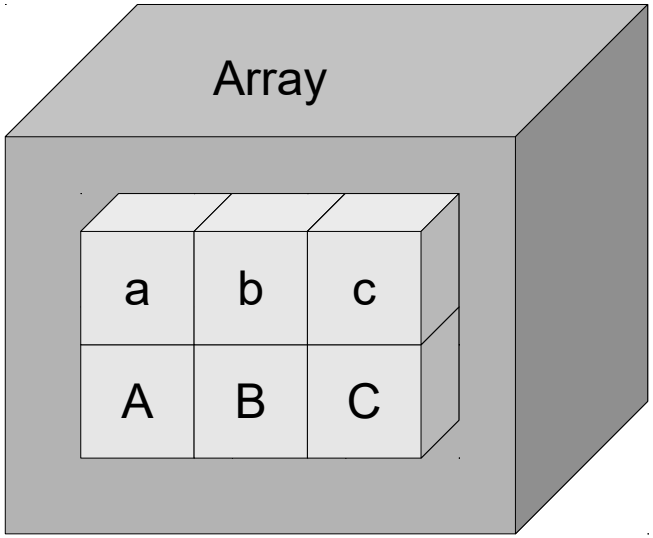
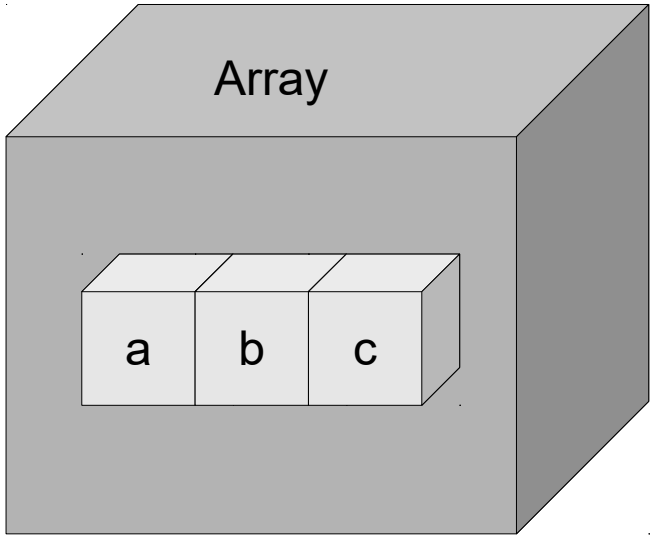
C++ - Einführung in die Programmiersprache

Arrays



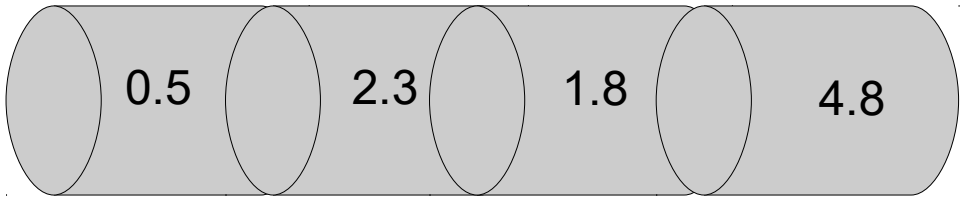


Array (Felder, Vektoren)

- Zusammenfassung von mehreren Variablen gleichen Datentyps.
- Gruppieren von Werten zu einem Thema. Zum Beispiel werden die täglichen Durchschnittstemperaturen eines Monats in einem Array gespeichert.
- Ein- oder mehrdimensionale Felder sind möglich.

Eindimensionale Arrays

- Folge von Werten gleichen Datentyps.
- Ein Container wird in verschiedene gleich große Boxen unterteilt. Jede Box hat einen bestimmten Inhalt.



Beispiel

- Aufzählung von Elementen wie zum Beispiel zu bestellende Waren, Namen der Monate und so weiter.
- Reihe von Zahlen wie zum Beispiel Temperaturwerte und so weiter.

Beispiele in C++

```
char buchstaben[26];
buchstaben[0] = 'a';
buchstaben[1] = 'b';

const int monatAnzahl = 12;
double temperaturen[monatAnzahl];
```

Deklaration von Variablen

`double`	`temperatur`		`;`
`Datentyp`	`Name`		`;`

- Jede Variable hat einen eindeutigen Namen. Dieser identifiziert den Speicherplatz des zu speichernden Wertes.
- Jede Variablen-Deklaration beginnt mit der Angabe eines Datentyps. Der Datentyp legt die Größe des Speicherplatzes fest. Die Art des zu speichernden Wertes wird bestimmt.
- Die Deklarationsanweisung endet mit einem Semikolon.

Deklaration von eindimensionalen Arrays

`char`	`buchstaben`	`[`	`26`	`]`	`;`
`Datentyp`	`Name`	`[`	`Anzahl`	`]`	`;`

- Jedes Array hat einen eindeutigen Namen. Der Name ist frei wählbar.
- Die Deklaration beginnt mit der Angabe eines Datentyps. Der Datentyp legt die Größe des Speicherplatzes eines Elements im Array fest. Die Art des zu speichernden Wertes wird bestimmt.
- In eckigen Klammern wird die Anzahl der Elemente angegeben. Die Anzahl kann als Literal oder mit Hilfe einer Variablen festgelegt werden.

Bezeichner

- Platzhalter für eine Konstante, Variable oder Array.
- Die Namen sind in ihrem Codeblock eindeutig.
- Schlüsselwörter der Programmiersprache sind als benutzerdefinierte Namen nicht erlaubt.
- Unterscheidung zwischen Groß- und Kleinschreibung. Die Namen „Temperaturen“ und „temperaturen“ sind unterschiedliche Platzhalter in C++.

Erlaubte Zeichen

- Buchstaben A...Z und a...z.
- Zahlen 0...9.
- Der Unterstrich.

Geeignete Bezeichner

- Der Name spiegelt die Nutzung des Bezeichners in dem Code wieder.
- Der Bezeichner nutzt den Namen des abgebildeten Objektes aus der Realität.
- Die Namen der Platzhalter nutzen eine Sprache. Die Sprachen englisch und deutsch, zum Beispiel, sollten nicht gemischt werden.
- Keine kryptischen Bezeichner wie a1, b etc. Bezeichner, die aus einem Buchstaben bestehen, sollten nur für Zähler in einer wiederholten Anweisung genutzt werden.

Namen von Arrays

- Häufig wird die Pluralform von Substantiven für Arrays genutzt.
- Der Name des Arrays verweist auf das erste Element im Array.

Angabe der Anzahl der Elemente

```
char buchstaben[26];
```

```
const int monatAnzahl = 12;
double temperatur[monatAnzahl];
```

… als Literal

```
char buchstaben[26];
```

- In den eckigen Klammern wird die Anzahl der Elemente des Arrays angegeben.
- Die Anzahl wird immer als Integer angegeben.
- Während der Programmausführung kann die Anzahl nicht erhöht oder vermindert werden.

… mit Hilfe einer Konstanten

```
const int monatAnzahl = 12;
double temperatur[monatAnzahl];
```

- In den eckigen Klammern wird die Anzahl der Elemente des Arrays angegeben.
- Die Anzahl kann durch eine Konstante vom Typ Integer angegeben werden. Die Konstante muss vor der Nutzung deklariert werden.
- Die Anzahl kann nur als konstanter Wert definiert werden. Während der Programmausführung kann die Anzahl nicht erhöht oder vermindert werden.

… deklarieren und initialisieren

`int zahlen[]`	`=`	`{1, 2, 3, 4}`	`;`
`Deklaration`	`=`	`Initialisierungsliste`	`;`

- Links vom Gleichheitszeichen wird ein Array deklariert.
- Mit Hilfe der Initialisierungsliste, rechts vom Gleichheitszeichen werden die einzelnen Elemente initialisiert.

Deklaration

`int zahlen[]`	`=`	`{1, 2, 3, 4}`	`;`
`Deklaration`	`=`	`Initialisierungsliste`	`;`

- Die Deklaration beginnt mit der Angabe eines Datentyps.
- Dem Datentyp folgt ein eindeutiger Name für das Array.
- Dem Namen folgen zwei leere eckige Klammern. Falls ein eindimensionales Array gleichzeitig deklariert und initialisiert wird, müssen diese leer sein.

Initialisierungsliste

`int zahlen[]`	`=`	`{1, 2, 3, 4}`	`;`
`Deklaration`	`=`	`Initialisierungsliste`	`;`

- Die Initialisierungsliste beginnt und endet mit den geschweiften Klammern.
- Die Liste hat x Elemente. Die Anzahl der Elemente in der Liste entspricht der Anzahl der Elemente in dem Array.
- Die Elemente in der Liste werden durch ein Komma getrennt.

… ab C++ 11

`int zahlen[]`	`=`	`{1, 2, 3, 4}`	`;`
`int zahlen[]`		`{1, 2, 3, 4}`	`;`

- Zwischen der Deklaration und der Initialisierungsliste wird kein Gleichheitszeichen benötigt.

Beispiel 1

```
int zahlen[ ] = {1, 2, 3, 4};

int werte[ ]{1, 2, 3, 4};
```

- Das Array besteht aus vier Elementen.
- Das erste Element des Array enthält den ersten Werten in der Liste, das zweite Element den zweiten Wert und so weiter.

Beispiel 2

```
double fieberkurve[20] = {};

double fieberkurve[20]{};
```

- Das Array hat zwanzig Elemente.
- Die Initialisierungsliste ist leer. Jedes Element in dem Array hat einen undefinierten Wert. Häufig wird 0 bei Zahlen und ein leeres Zeichen für den Datentyp `char` genutzt.

Beispiel 3

```
int mehrZahlen[]{};

int vieleZahlen[] = {};
```

- Das Array enthält keine Elemente. Das Array hat eine Länge von 0.
- Die Elemente in dem Array verweisen auf einen x beliebigen Wert.

Anzahl der Elemente

```
int werte[ ]{1, 2, 3, 4};
int anzahl = 0;

anzahl = sizeof(werte) / sizeof(int);
anzahl = sizeof(werte) / sizeof(werte[0]);
```

- Es ist keine Funktion in der Standardbibliothek von C++ zur Berechnung vorhanden.
- Die Größe in Bytes des Arrays wird durch die Größe in Bytes eines Elements dividiert.

sizeof()

```
int werte[ ]{1, 2, 3, 4};
int anzahl = 0;

anzahl = sizeof(werte) / sizeof(int);
anzahl = sizeof(werte) / sizeof(werte[0]);
```

- Die Funktion `sizeof()` gibt die Größe eines beliebigen Elements in Bytes zurück.
- Dem Namen der Funktion folgen die runden Klammern. Die Parameterliste hat einen Parameter.
- Parameter der Funktion: Für welches Element wird die Größe bestimmt? Der Speicherbedarf in Bytes für dieses Element wird ermittelt.

Erläuterung

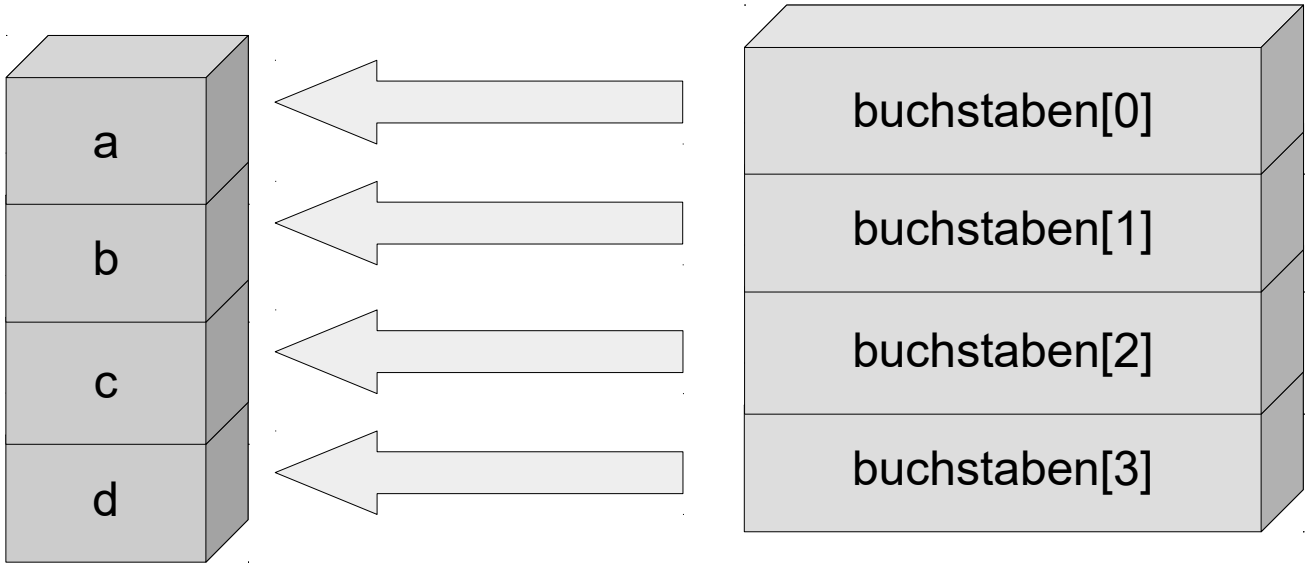
- Mit Hilfe von `sizeof(werte)` wird die Größe in Bytes eines Arrays berechnet.
- Mit Hilfe von `sizeof(int)` wird die Größe in Bytes eines Integer-Wertes berechnet. Wie viel Speicherbedarf hat ein Integer-Wert.
- Mit Hilfe von `sizeof(werte[0])` wird die Größe eines bestimmten Elements im Array berechnet. In diesem Beispiel wird die Größe des ersten Elements ermittelt.

Elemente des Arrays

```
buchstaben[0] = 'a';
cout << buchstaben[0];
```

- Elemente eines Arrays werden in einer Ausdrucksanweisung genutzt.
- Dem Namen des Arrays folgt der Index in eckigen Klammern.
- Der Index ist eine Ganzzahl. Das erste Element hat die 0, das zweite Element die 1 und so weiter.
- Der Index ist eindeutig. Mit Hilfe des Index wird ein Element in einem Array exakt identifiziert.

Beispiel



Index als Literal angeben

```
buchstaben[0] = 'a';
cout << buchstaben[0];
```

- Eine Ganzzahl kann direkt in die eckigen Klammern eingegeben werden.
- In diesem Beispiel wird das erste Element ausgegeben.

Index als Variable angeben

```
int anzahlElement = sizeof(buchstaben) / sizeof(char);

int index = anzahlElement - 1;
buchstaben[index] = 'z';
std::cout << "Index " << index << ": " << buchstaben[index];
```

- Eine Variable kann auch als Index genutzt werden.
- Statt einer Ganzzahl wird der Variablennamen in die eckigen Klammern gesetzt.
- In diesem Beispiel wird das letzte Element ausgegeben.

Ausgabe eines Arrays (vorwärts)

```
char gen[] = {'G', 'A', 'T', 'C', 'A', 'A', 'G', 'G', 'G'};

int anzahlElement = sizeof(gen) / sizeof(char);

for(int index = 0; index < anzahlElement; index++ )
{
    std::cout << " - " << gen[index];
}
```

Ausgabe eines Arrays (rückwärts)

```
char gen[] = {'G', 'A', 'T', 'C', 'A', 'A', 'G', 'G', 'G'};

int anzahlElement = sizeof(gen) / sizeof(char);
int indexLast = anzahlElement - 1;

for(int index = indexLast; index >= 0; index-- )
{
   std::cout << " - " << gen[index];
}
```

Kopieren von Arrays

```
char gen[] = {'G', 'A', 'T', 'C', 'A', 'A', 'G', 'G', 'G'};

int anzahlElement = sizeof(gen) / sizeof(char);

char sequenz[anzahlElement];


for(int index = 0; index < anzahlElement; index++ )
{
    sequenz[index] = gen[index];

}
```

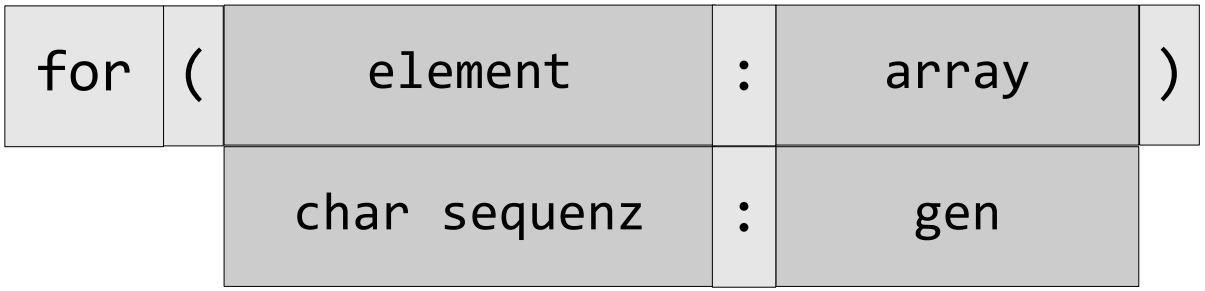
foreach-Schleife

```
char gen[] = {'G', 'A', 'T', 'C', 'A', 'A', 'G', 'G', 'G'};

for(char element : gen)
{
    std::cout << " - " << element;
}
```

Schleifenkopf

- Der Schleifenkopf beginnt mit dem Schlüsselwort `for`.
- In den runden Klammern wird definiert, welches Array vollständig durchlaufen werden soll.

Für jedes Element



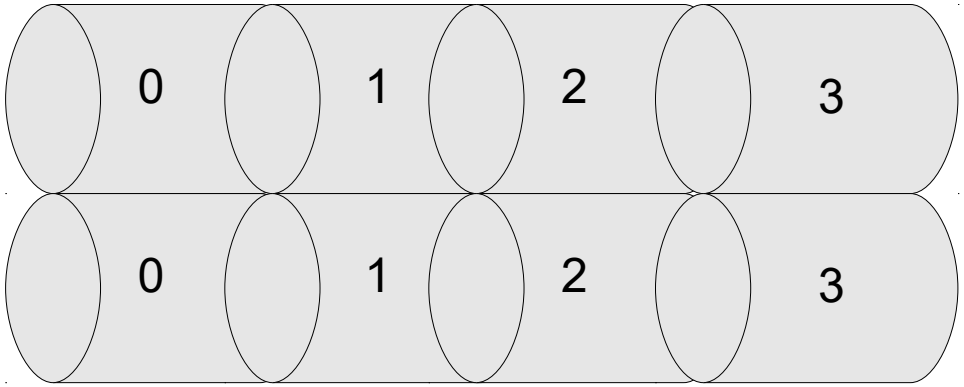
- Links vom Doppelpunkt wird eine Variable definiert. Die Variable symbolisiert ein Element in einem Array.
- Rechts vom Doppelpunkt wird das Array angegeben.
- Die Variable und das Array sind vom gleichen Datentyp.

Hinweise

- Das Array wird vollständig vom ersten bis zum letzten Element durchlaufen.
- Die Array-Elemente können nicht verändert werden. Die Werte des Array-Elements werden an die Variable „call by value“ automatisiert übergeben.

Mehrdimensionale Arrays

- Darstellung von mehreren Dimensionen.
- Zwei Dimensionen: x-, y-Koordinatensystem; Schachbrett; Matrizen.
- Drei Dimensionen: x-, y-, z-Koordinatensystem.



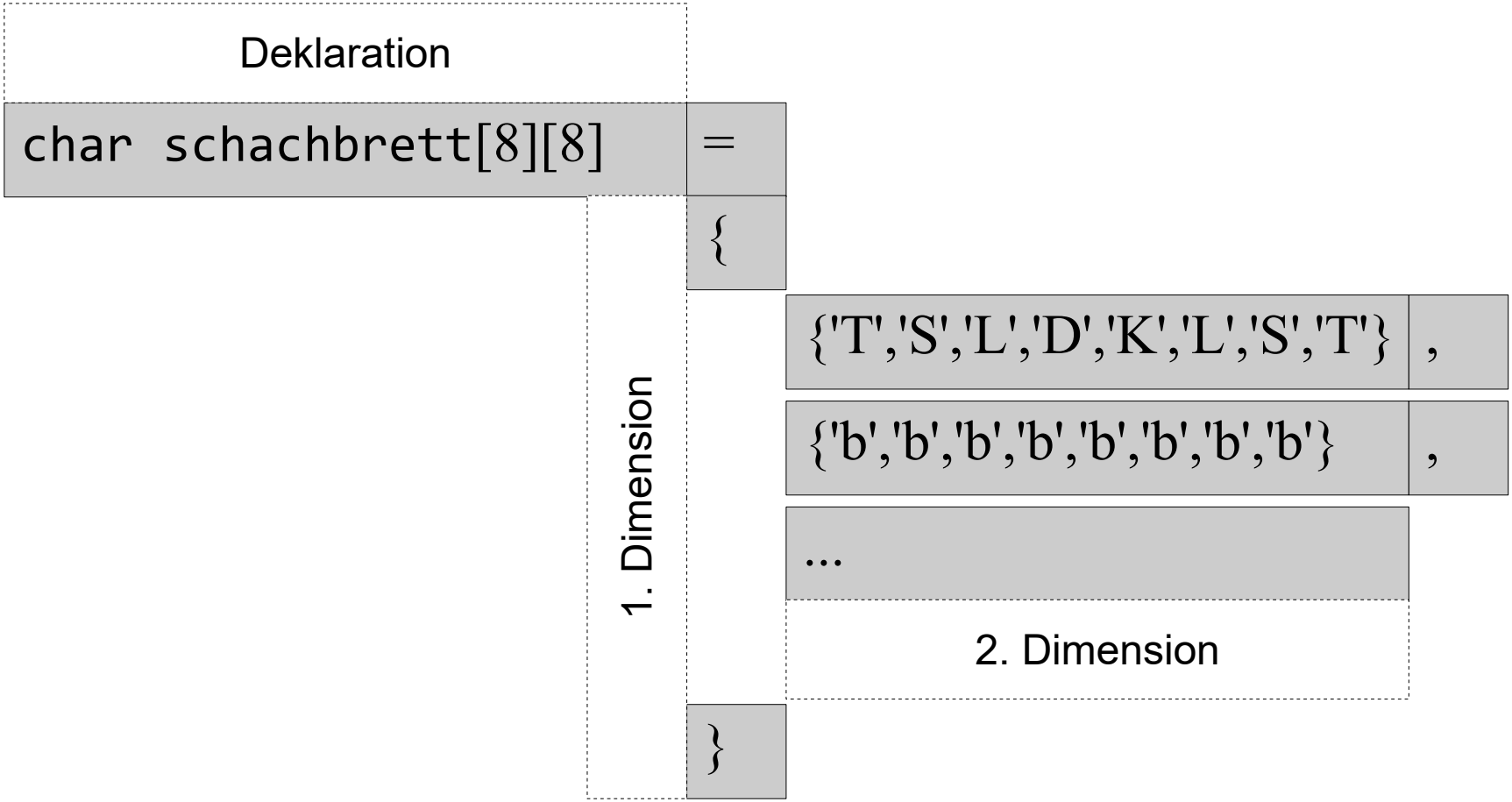
… in C++

```
const int anzahlFelder  = 8;
char schachbrett[anzahlFelder][anzahlFelder];

double monatTage[12][31];

int k[][3][4]  = {
                    {
                       {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                    },

                    {
                       {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                    }
                };
```

Hinweise

- Die maximal mögliche Dimension ist abhängig vom verwendeten Compiler.
- Der benötigte Speicherplatz steigt exponentiell mit der Anzahl der Dimensionen.

Beispiel: Zweidimensionales Array



Deklaration von mehrdimensionalen Arrays

`double`	`monatTag`	`[`	`12`	`]`	`[`	`31`	`]`
`Datentyp`	`Name`	`[`	`Dimension`	`]`	`[`	`Dimension`	`]`

- Jedes Array hat einen eindeutigen Namen. Der Name ist frei wählbar.
- Die Deklaration beginnt mit der Angabe eines Datentyps. Der Datentyp legt die Größe des Speicherplatzes eines Elements im Array fest.
- In eckigen Klammern wird die Anzahl der Elemente für jede Dimension angegeben. In diesem Beispiel wird ein zweidimensionales Array deklariert.

Angabe der Anzahl der Elemente

```
double monatTage[12][31];
```

```
const int anzahlFelder  = 8;
char schachbrett[anzahlFelder][anzahlFelder];
```

… als Literal

```
double monatTage[12][31];
```

- In den eckigen Klammern wird die Anzahl der Elemente für jede Dimension angegeben
- Die Anzahl wird immer als Integer angegeben.a
- Während der Programmausführung kann die Anzahl nicht erhöht oder vermindert werden.

… mit Hilfe einer Konstanten

```
const int anzahlFelder  = 8;
char schachbrett[anzahlFelder][anzahlFelder];
```

- In den eckigen Klammern wird die Anzahl der Elemente für jede Dimension des Arrays angegeben.
- Die Anzahl kann durch eine Konstante vom Typ Integer angegeben werden. Die Konstante muss vor der Nutzung deklariert werden.
- Die Anzahl kann nur als konstanter Wert definiert werden. Während der Programmausführung kann die Anzahl nicht erhöht oder vermindert werden.

Deklaration und Initialisierung

```
int k[][3][4]  = {
                  {
                   {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                  },
                  {
                   {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                  }
                 };
```

Deklaration

`int k[][3][4]`	`=`	`{ }`	`;`
`Deklaration`	`=`	`Initialisierungsliste`	`;`

- Die eckigen Klammern für die erste Dimension können leer sein. Die Anzahl der Elemente für die erste Dimension müssen nicht angegeben werden.
- Für alle anderen Dimensionen muss die Anzahl der Elemente angegeben werden.

Initialisierungsliste

```
int k[][3][4]  = {
                  {
                   {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                  },
                  {
                   {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                  }
                 };
```

- Jede Initialisierungsliste beginnt und endet mit den geschweiften Klammern.

Initialisierungsliste

```
int k[][3][4]  = {
    {
      {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
    },
    {
      {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
    }
};
```

2. Dimension

3. Dimension

1. Dimension

… für die Dimension

```
int k[][3][4]  = {
                  {
                   {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                  },
                  {
                   {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                  }
                 };
```

- Die Initialisierungsliste der ersten Dimension besteht aus den Initialisierungslisten der zweiten Dimension und so weiter.
- Die Listen werden in einander verschachtelt.
- Die einzelnen Listen werden durch ein Komma getrennt.

Werte des mehrdimensionalen Arrays

```
int k[][3][4]  = {
                   {
                    {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                   },
                   {
                    {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                   }
                  };
```

- In der letzten Dimension werden die Werte des Arrays eingetragen.
- Die Werte in der Initialisierungsliste werden durch ein Komma getrennt.

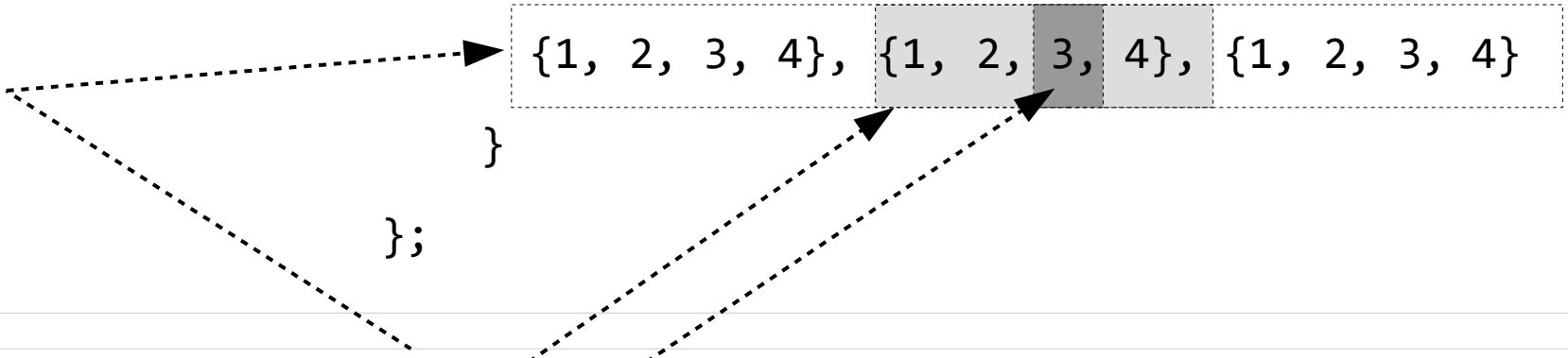
Elemente in einem mehrdimensionalen Array

```
double monatTage[12][31];

monatTage[1][3] = 2.8;

cout << monatTage[1][3];
```

- Für jede Dimension wird ein Index angegeben.
- Jeder Index identifiziert eindeutig ein Element in der Dimension.

Beispiel

```
int k[][3][4]  = {
                    {
                      {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                    },
                    {
                      {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}
                    }
                };
```



```
std::cout << k[1][1][2];
```

Größe eines mehrdimensionalen Arrays

```
double monatTage[12][31];

monatTage[0][0]  = 0.1;

std::cout << "\n Groesse des Arrays: " << sizeof(monatTage);
std::cout << "\n Anzahl gesamt: "
          << (sizeof(monatTage) / sizeof(double));

std::cout << "\n Groesse 2. Dimension: " << sizeof(monatTage[0]);
std::cout << "\n Anzahl 1. Dimension: "
          << (sizeof(monatTage) / sizeof(monatTage[0]));

std::cout << "\n Groesse 2. Dimension: "
          << sizeof(monatTage[0][0]);
std::cout << "\n Anzahl 2. Dimension: "
          << (sizeof(monatTage[0]) / sizeof(monatTage[0][0]));
```

… einlesen

```
double monatTage[12][31];
double temperatur;

int anzahl = sizeof(monatTage) / sizeof(double);
int anzahl2Dim = sizeof(monatTage[0]) / sizeof(double);
int anzahl1Dim = sizeof(monatTage) / sizeof(monatTage[0]);

for(int monatNr = 0; monatNr < anzahl1Dim; monatNr++){
    for (int tagNr = 0; tagNr < anzahl2Dim; tagNr++){

        cout << "\nTemperatur am " << (tagNr + 1) << ". "
             << (monatNr + 1) << ". ein: ";

            cin >> monatTage[monatNr][tagNr];
    }

}
```