Arrays

>> Arrays sind eine sehr nützliche und fundamentale Datenstruktur. Sie speichern Werte des gleichen Datentyps an zusammenhängenden Speicherorten.

Wie die *Postfächer in einer Postfiliale:*

Große Wand mit vielen gleich großen Fächern. Jedes Fach kann einen bestimmten Inhalt aufnehmen und es ist ein direkter Zugriff über eine Nummer möglich.


41
61
42
62
43
63
44
64
45
65
46
66
47
67

Arrays	Postfächer
Block zusammen- hängenden Speichers	Große Wand in der Postfiliale
Unterteilt in Elemente	Unterteilt in Postfächer
Jedes Element speichert bestimmte Datenmenge	Jedes Fach enthält bestimmte Postmenge
Alle Elemente vom gleichen Datentyp	Verschiedene Post- arten möglich
Zugriff auf Elemente über Index	Zugriff auf Inhalt über Fachnummer
Indizierung beginnt bei 0	Nummerierung beginnt bei 1

Arrays in C

Die Indizierung beginnt bei 0.

In Schleifen beginnt die Zählung daher üblicherweise bei 0.

Array mit n Elementen:

- Erstes Element hat Index 0.
- Letztes Element hat Index $n-1$.

Zugriffe auf höhere Indizes als $n-1$ werden von C grundsätzlich nicht verhindert – können zu Fehlern oder Abstürzen führen („out of bounds“).

Tipp: Stellen Sie sich den Index als „Anzahl der Schritte vom Anfang“ vor.

Deklaration und Zugriff

```
type name[size];
```

type: Datentyp jedes Elements (z.B. int oder double)

name: Bezeichnung des Arrays

size: Anzahl der Elemente

Beispiel für Deklaration:

```
int student_grades[40];
```

```
double menu_prices[8];
```

Beispiel für Zugriff:

```
student_grades[2] = 95;
```

```
if(menu_prices[0] > 10.0) { ... }
```

Initialisierung

Spezielle Syntax bei gleichzeitiger Deklaration und Initialisierung:

```
typ name[] = { wert1, wert2, ... };
```

Beispiel:

```
bool truthtable[3] = { false, true, true };
```

Äquivalent zu:

```
bool truthtable[] = { false, true, true };
```

Zugriff auf Elemente:

```
truthtable[0] = true;
```

```
if (truthtable[2] == true) { ... }
```

Mehrdimensionale Arrays

```
typ name[size1][size2]...;
```

Beispiel „Schiffe versenken“:

```
bool battleship[10][10];
```

Erzeugt ein 10x10 Spielfeld (2D-Matrix);

Intern: 100 Elemente in Reihe.

Beispiel für den Zugriff:

```
battleship[0][9] = true;
```

Es gilt:

```
battleship[1][0] == battleship[0][10]
```

(Der Compiler von CS50 wertet dies als Fehler.)

Zuweisungen

Einzelne Array-Elemente können wie Variablen behandelt werden.

Gesamte Arrays können aber nicht direkt zugewiesen werden:

```
int foo[5] = {1, 2, 3, 4, 5};
int bar[5];
bar = foo; // Fehler!
```

Korrekte Methode (vorerst):

```
for (int j = 0; j < 5; j++)
{
    bar[j] = foo[j];
}
```

Mehr Kontrolle, Vermeidung unbeabsichtigter aufwändiger Kopiervorgänge.

Übergabe als Referenz

Erinnern Sie sich daran, dass Variablen in C bei Funktionsaufrufen üblicherweise **als Wert** übergeben werden.

Bei Arrays gilt diese Regel nicht. Sie werden **als Referenz** übergeben. Die aufgerufene Funktion (*Callee*) erhält das eigentliche Array, nicht eine Kopie davon.

Was bedeutet das, wenn der Callee Elemente im Array verändert?

Warum Arrays diese besondere Eigenschaft haben, werden wir vorerst nicht weiter ausführen – aber wir werden demnächst darauf zurückkommen.

Beispiel

```c
void set_int(int x) { x = 22; }
void set_array(int array[4]) { array[0] = 22; }

int main(void)
{
    int a = 10;
    int b[4] = {0, 1, 2, 3};
    set_int(a);
    set_array(b);
    printf("%d %d\n", a, b[0]);
}
```

Ausgabe: `10 22`

Begründung:

- a bleibt 10 (Übergabe als Wert)
- b[0] wird zu 22 (Übergabe als Referenz)

EXTRAS IN 3 MINUTEN

FRAGEN – ANTWORTEN – RÄTSEL
UND KURZE ZUSAMMENFASSUNG

F1: Warum beginnt die Indizierung von Arrays in C bei 0 und nicht bei 1?

F2: Wie würden Sie ein zweidimensionales Array für ein Schachbrett deklarieren?

F3: Welcher Unterschied besteht zwischen der Übergabe eines einzelnen Integers und eines Integer-Arrays an eine Funktion?

```c
int main() {
    int zahlen[5] = {1, 2, 3, 4, 5};
    for(int i = 0; i <= 5; i++) {
        printf("%d ", zahlen[i]);
    }
    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    int arr[2][3];
    // fill array (arr[0][0] = 0 ... arr[1][2] = 5)
    for (int i = 0, counter = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            arr[i][j] = counter++;
        }
    }
    for (int i = 0; i < 2 * 3; i++)
    {
        printf("[%d] = %d\n", i, arr[0][i]);
    }
}
```

Arrays sind zusammenhängende Speicherblöcke für Elemente gleichen Typs

Initialisierung:

typ name[] = {wert1, wert2, ...};

Indizierung beginnt in C bei 0, bei *n* Elementen ist der letzte Index also (*n*–1)

Mehrdimensionale Arrays werden im Speicher linear angeordnet

Deklaration: typ name[größe];

z.B. int zahlen[5];

Arrays werden an Funktionen „als Referenz“ übergeben, nicht „als Wert“