2D Arrays

Sebastian Stabinger, Thomas Hausberger

SS2021

2D Arrays im Speicher

Das Problem mit 2D Arrays

- Wir würden gerne 2D Arrays (z.B. Matrizen) in C verwenden.
- Das Problem: Speicher ist üblicherweise (bis auf sehr seltene Spezialhardware) eindimensional
- Wir müssen also eine Möglichkeit finden um ein 2D Array mit Hilfe eines 1D Arrays zu simulieren
- Wir besprechen das Problem und die Lösung an Hand von 2D Arrays. Das Problem und die Lösungen gelten aber genauso für alle höherdimensionalen Arrays (z.B. 3D Tensoren)

2D-Arrays in C implementiert?

Zwei Möglichkeiten

- In C integrierte mehrdimensionale Arrays
- Manuelle Lösung

In C integrierte Version

Syntax:

```
datentyp name[Zeilenanzahl][Spaltenanzahl];
```

Beispiel

```
int a[3][4];
a[1][2] = 23;
a[0][3] = 42;
printf("Wert in Zeile 2, Spalte 3 = %d", a[1][2]);
```

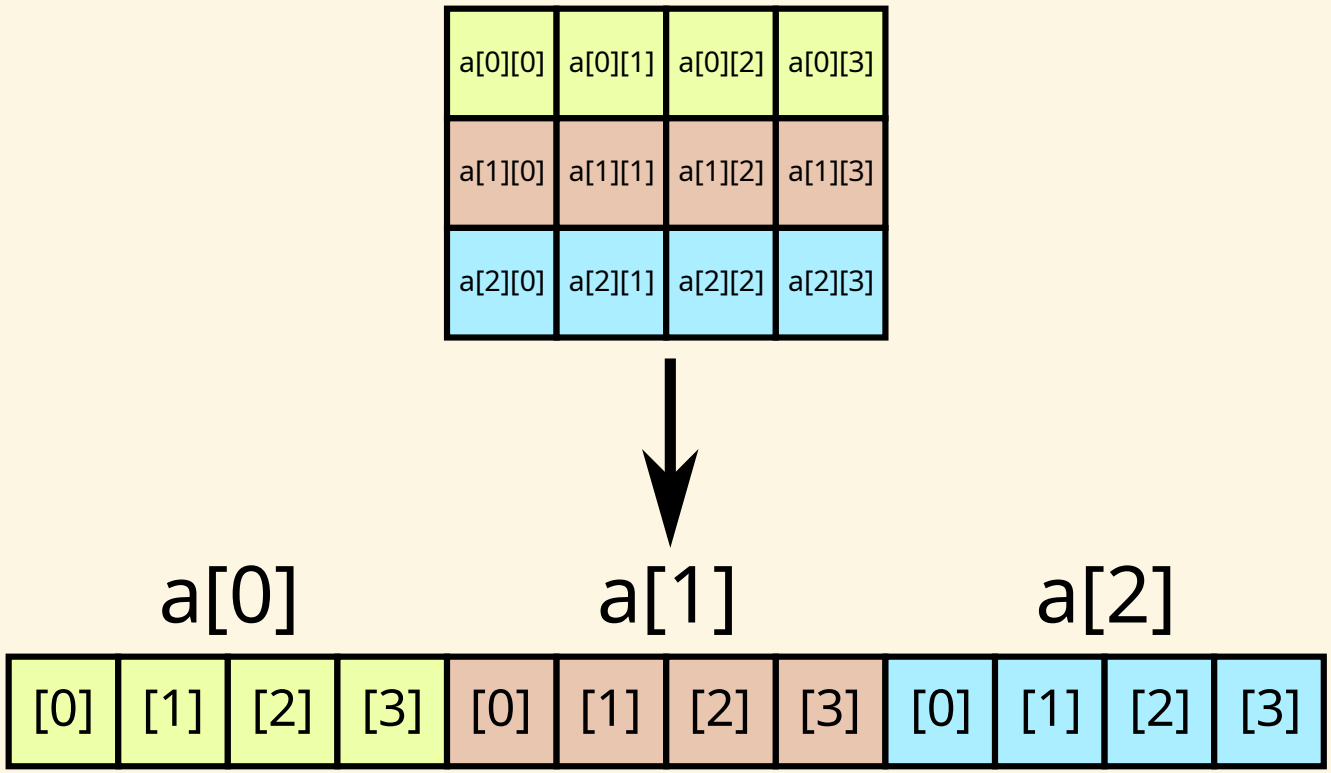
Es sind auch mehr als zwei Dimensionen möglich

```
// 4D Array
double arr[12][34][42][3];
```

Visualisierung

Was passiert im Speicher bei `int a[3][4];`?

- Man reserviert Speicher für ein Array mit 3 Elementen, wobei jedes Element ein Array mit vier Integer Werten ist!



- `a[1]` liefert also z.B. das Array für die zweite Zeile des Arrays zurück auf dessen drittes Elemente man dann mittels `[2]` zugreift

Initialisieren

Verschachtelte Initialisierungslisten

```
int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
```

Manuell

```
for (int r = 0; r < 3; r++)
  for (int c = 0; c < 4; c++)
    a[r][c] = 0;
```

Achtung: Die innerste Schleife sollte ueber die letzte Dimension iterieren, da dies schneller ist!

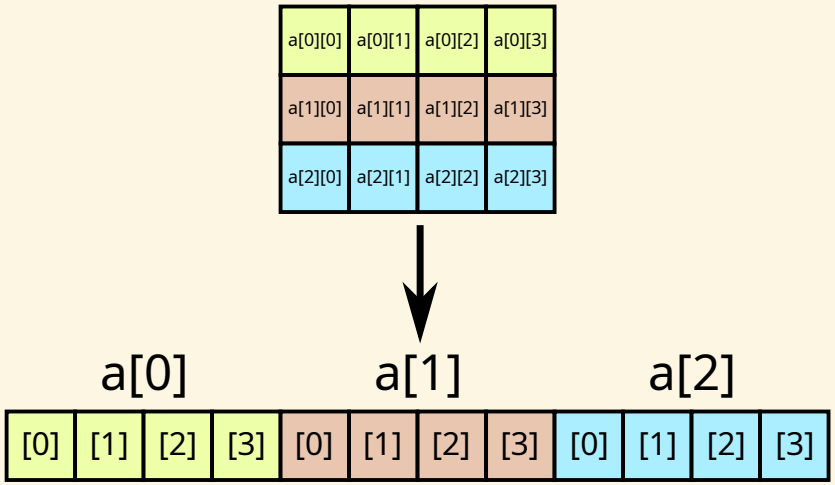
In C integrierte Version — Beispiel

Beispiel

```c
#include <stdio.h>

int main() {
  int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};

  for (int row = 0; row < 3; row++) {
    for (int col = 0; col < 4; col++)
      printf("%d\t", a[row][col]);
    printf("\n");
  }
}
```

Übergabe an Funktionen

Um auf Werte eines 2D Arrays zugreifen zu können, muss C wissen wie viele Werte eine Zeile enthält!



→ Wenn ein mehrdimensionales Array an eine Funktion übergeben wird muss die Größe aller Dimensionen bis auf die erste bekannt sein!

Übergabe an Funktionen — Beispiel

Beispiel

```c
#include <stdio.h>

void print_matrix(int arr[][4], int rows) {
  for (int row = 0; row < rows; row++) {
    for (int col = 0; col < 4; col++)
      printf("%d\t", arr[row][col]);
    printf("\n");
  }
}

int main() {
  int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
  print_matrix(a, 3);
}
```

Übergabe an Funktionen — Beispiel seit C99

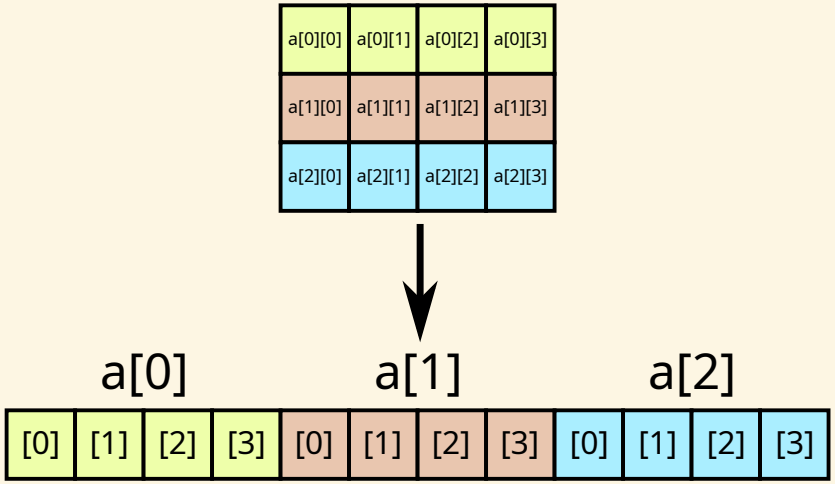
- Seit C99 können übergebene Parameter statt fixen Größen verwendet werden

Beispiel

```c
#include <stdio.h>

void print_matrix(int rows, int cols, int arr[][cols]) {
  for (int row = 0; row < rows; row++) {
    for (int col = 0; col < cols; col++)
      printf("%d\t", arr[row][col]);
    printf("\n");
  }
}

int main() {
  int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
  print_matrix(3, 4, a);
}
```

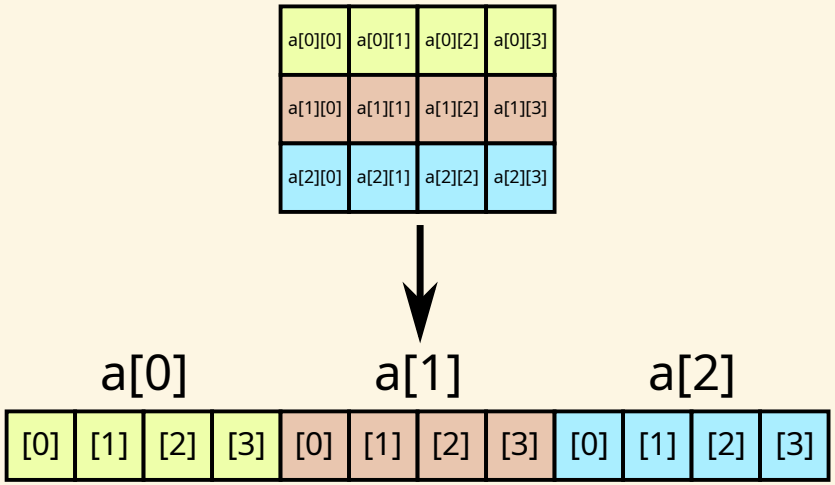
Manuell

- Das was C intern macht, kann man auch einfach manuell machen
- Man erzeugt ein 1D-Array der Größe: Zeilenanzahl * Spaltenanzahl



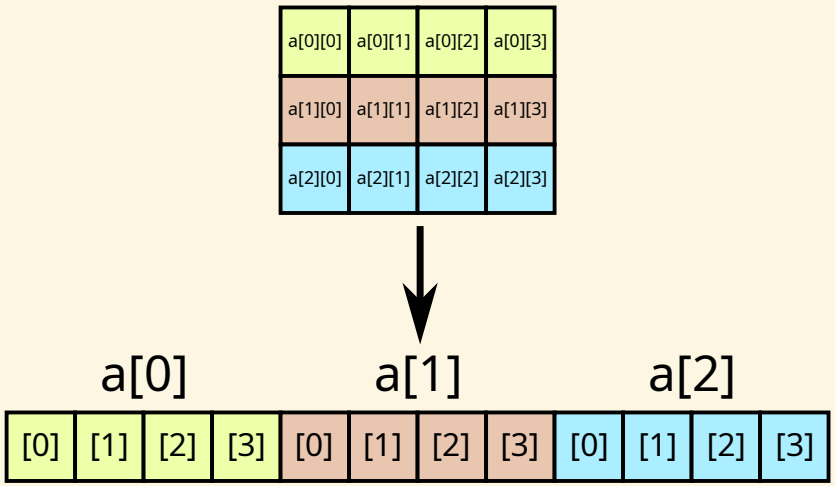
- Achtung: Diese manuelle Methode ist NICHT langsamer als das was C intern macht!
- Ich bevorzuge diese Methode, da sie meiner Meinung nach einfacher und flexibler ist

Manuell — Zeilenanfang ermitteln



- Um auf ein Element in einer gewissen Zeile zuzugreifen müssen wir also berechnen wo diese Zeile im Array anfängt. Wo die Zeile anfängt hängt davon ab, wie lang eine Zeile ist (also die Anzahl an Spalten)
  - Anfang der Zeile = Zeilennummer $\times$ Anzahl an Spalten
  - z.B. Anfang der Zeile $2 = 2 \times 4 = 8$

Manuell — Berücksichtigen der Spalte



- Wir wissen jetzt an welcher Position eine Zeile anfängt. Wenn wir ein Element in dieser Zeile in einer bestimmten Spalte wollen addieren wir zum Zeilenanfang die Spaltennummer
  - Position von Element = Zeilennummer $\times$ Anzahl an Spalten + Spaltennummer
  - z.B. Position von Element in Zeile 1 und Spalte 2 = $1 \times 4 + 2 = 6$

Manuell — Beispiel

Beispiel

```c
#include <stdio.h>

int main() {
  int rows = 3;
  int cols = 4;
  int a[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

  for (int row = 0; row < 3; row++) {
    for (int col = 0; col < 4; col++)
      printf("%d\t", a[row * cols + col]);
    printf("\n");
  }
}
```

Manuell — Beispiel — Übergabe an Funktionen

Beispiel

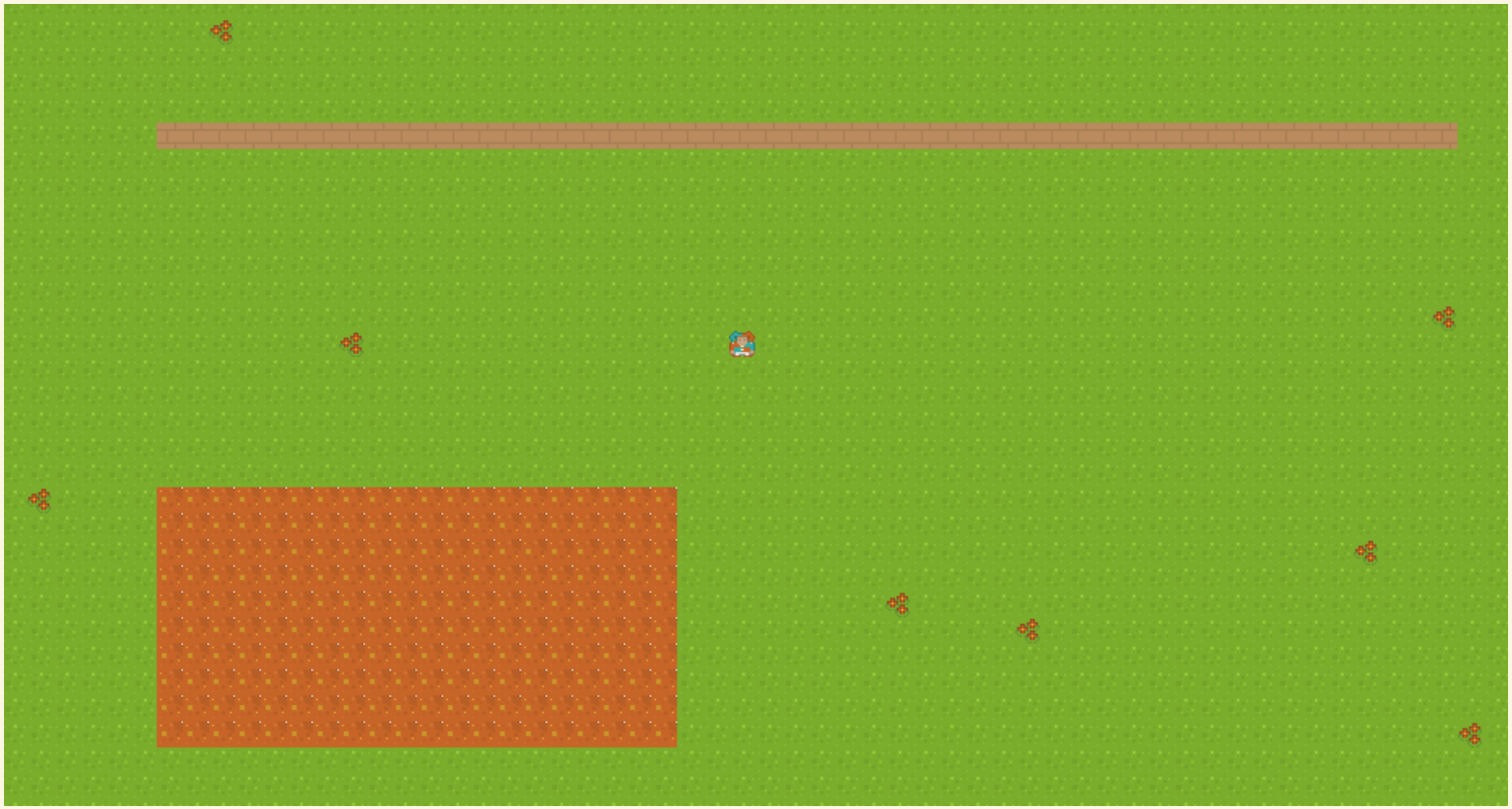
```c
#include <stdio.h>

void print_matrix(int *arr, int rows, int cols) {
  for (int row = 0; row < rows; row++) {
    for (int col = 0; col < cols; col++)
      printf("%d\t", arr[row * cols + col]);
    printf("\n");
  }
}

int main() {
  int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
  print_matrix(a, 3, 4);
}
```

Dynamische Speicherverwaltung

- Man kann mittels dynamischer Speicherverwaltung mehrdimensionale Arrays erzeugen die sich verhalten wie die von C intern unterstützten. Das ist aber relativ umständlich und kompliziert
- Meine Empfehlung: Verwenden der manuellen Methode

Gemeinsames Beispiel

Wir wollen in einem 2D Array speichern was an einer bestimmten Position als Hintergrund gezeichnet werden soll



Übung

- Erweitern Sie das vorherige Beispiel so, dass Sie in einem zweiten 2D Array speichern welche Felder begehbar sind und welche nicht
- Übergeben Sie dieses Array an die Bewegungsfunktionen (`move_left`, `move_right`, `move_up`, `move_down`) und verhindern Sie in diesen, dass unsere Spielfigur Felder betreten kann welche als nicht begehbar markiert sind.
- Machen Sie damit unsere gezeichnete Mauer unpassierbar