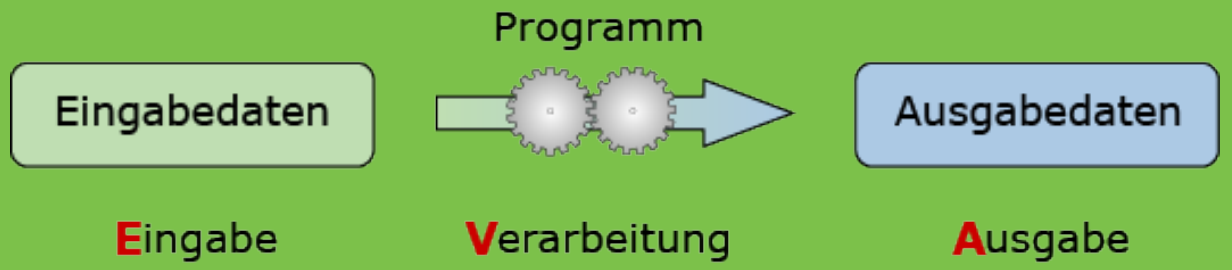
Hochschule Fulda
University of Applied Sciences

Programmierung 1 – Arrays und C-Strings



Yvonne Jung

Felder (Arrays)

Hochschule Fulda
University of Applied Sciences

- Oft müssen viele Werte desselben Datentyps gespeichert werden
  - Hier ist es praktisch, diese Werte konsekutiv im Speicher abzulegen
  - Dabei wird über einen Index auf die einzelnen Werte zugegriffen

0	1	2	3	4	5	6	7	*Index*
0	**1**	**1**	**2**	**3**	**5**	**8**	**13**	***Inhalt***

- Felder enthalten immer nur Elemente des *gleichen* Datentyps
  - Die Anzahl der Elemente ist unveränderbar (Felder haben feste Länge)!
  - Die Elemente an den einzelnen Indizes (Positionen) sind aber änderbar
  - Indexzugriff erlaubt wahlfreien Zugriff auf einzelne Feldelemente

Erzeugen von Arrays

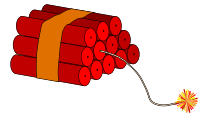
Hochschule Fulda
University of Applied Sciences

- Feldlänge wird bei Variablendeklaration in eckigen Klammern angegeben und an Bezeichner angehängt
  - Bsp. 1 (Integer-Array mit zehn Elementen): `int werte[10];`
    - Hinweis: Vor C99 (bzw. bei älteren Compilern) sowie außerhalb von Funktionen muss Feldlänge *konstanter* Ausdruck sein (wäre sowohl bei Bsp. 1 als auch bei Bsp. 2 gegeben)
  - Bsp. 2 (Char-Array): `char str[] = {'p','r','o','g','\0'};`
    - Hier wird Feld explizit mit Mengenschreibweise unter Angabe aller Elemente deklariert und initialisiert; Feldlänge (hier 5) ergibt sich automatisch aus Anzahl der Elemente
- Achtung: in C kennt Feld seine Länge nicht!
  - Feldlänge muss zusätzlich (als Konstante / Variable) abgespeichert werden

Elementzugriff

Hochschule Fulda
University of Applied Sciences

- Indizes gehen bei einer Feldlänge von *n* von 0 bis *n*-1
- Auf einzelne Feldposition greift man zu, indem man Integer-Ausdruck in eckige Klammern hinter Feldnamen schreibt
- Überschreiten der Feldgrenzen ist Fehler, wird in C aber *nicht* überprüft!



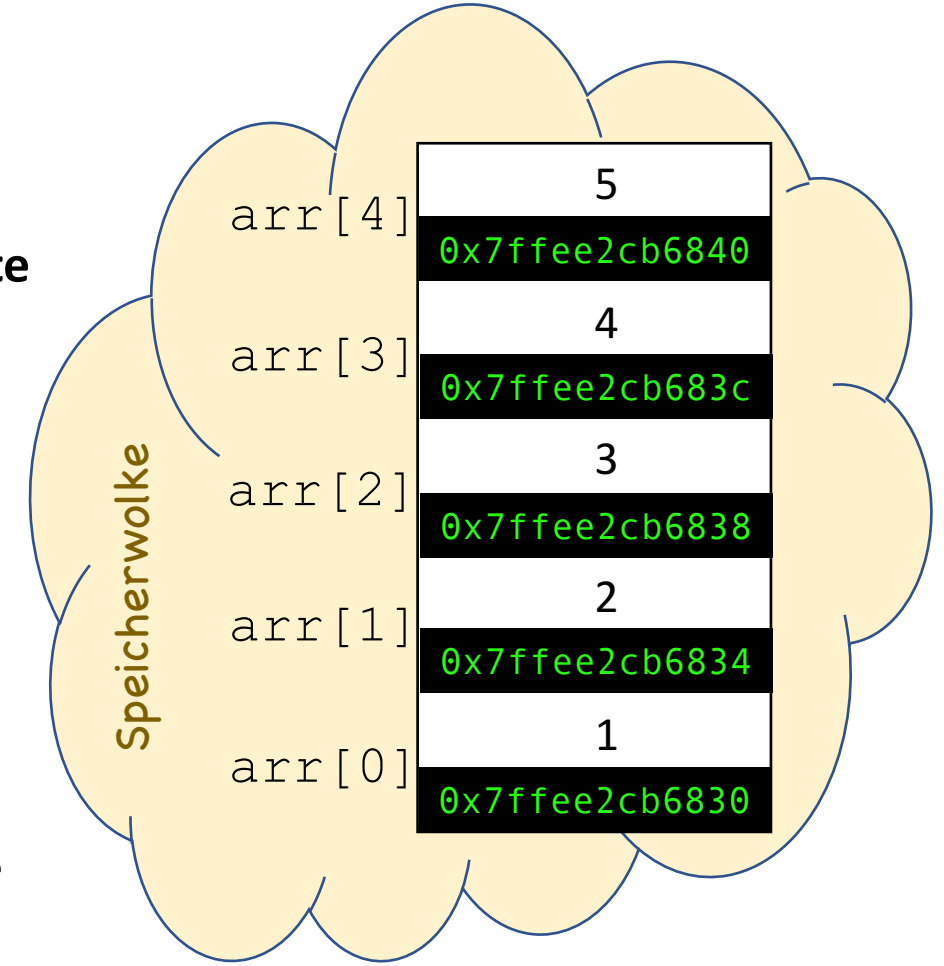
```c
const int N = 10;
// Deklaration eines Feldes der Länge N
int feld[N];
// Initialisierung aller Feldwerte mit 0
for (int i=0; i<N; i++) {
    feld[i] = 0;
}
// Zugriff auf 5. Element (Startindex 0)
printf("feld[%d] ist %d\n", 4, feld[4]);
```

Arrays im Speicher

Hochschule Fulda
University of Applied Sciences

```
int arr[] = {1, 2, 3, 4, 5};
```

- Felder sind Adresskonstanten
  - Feldname (ohne Index dahinter) enthält als Wert **konstante** Anfangsadresse des Feldes (Adresse des ersten Elements)
  - Felder kann man deshalb in C einander *nicht* zuweisen
  - Inhalte verschiedener Felder kann man *nicht* mit '==' vergleichen, da man so nur Anfangsadressen vergleicht
- Felder sind zusammenhängende Speicherbereiche
  - Ein Array zu indizieren bedeutet, intern einen Offset auf die Adresse des ersten Feldelements zu addieren
  - Größe des Offsets abhängig vom Typ, bei `int` i.d.R. 4 Byte



Übung

Hochschule Fulda
University of Applied Sciences

- Geben Sie an, wie ein Integer-Array der Länge 100 deklariert wird
- Deklarieren Sie ein Float-Array und initialisieren Sie es direkt mit Beispielwerten
- Warum sollte die Länge eines Arrays (z.B. *arr[N]*) mit Hilfe einer zuvor definierten Konstanten (z.B. *N*) deklariert werden?
- Was wird ausgegeben?

```
int k = 0, arr1[] = {3, 4, 5}, arr2[3];
arr2[0] = 3;
arr2[1] = 4;
arr2[2] = 5;
printf("%s\n", arr1 == arr2 ? "true" : "false");
arr1[++k] = 7;
++arr2[k++];
printf("%d, %d\n", arr1[1], arr2[1]);
```

Felder als Funktionsparameter

Hochschule Fulda
University of Applied Sciences

```
int sumOfArray(int values[], int numValues);
```

Array in Parameterliste mit "[]" gekennzeichnet

- Aufgerufene Funktion kennt Länge des übergebenen Arrays nicht
  - Da Arrays ihre Länge ja selbst nicht kennen…
  - Feldlänge muss daher separat als weiterer Parameter übergeben werden
- Feldinhalte werden bei Funktionsaufruf nicht kopiert
  - Lediglich der Wert der Feldvariablen (also Adresse des ersten Feldelements) wird bei Aufruf an Funktion übergeben

```
int ret = sumOfArray(feld, N);    // Zuvor: int feld[N];
```

  - Verändert man Feld in Funktion, so ist auch übergebenes Feld in aufrufender Funktion verändert, weil Feldvariable je auf gleiche Start-Speicherstelle zeigt

Beispiel: Tic Tac Toe

Hochschule Fulda
University of Applied Sciences

- Spielfeld der Größe 3 x 3
- Zwei Spieler, die abwechselnd Steine setzen
- Wer zuerst drei Steine in einer Reihe hat, gewinnt

	O	
O	X	X

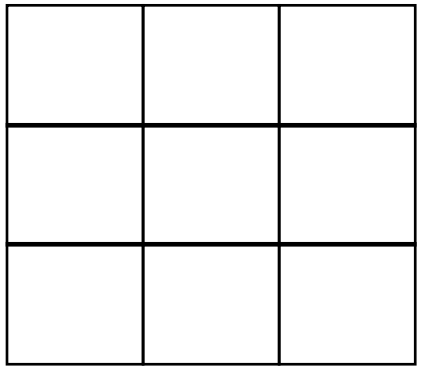
- Jede Reihe könnte man als Feld mit Länge 3 anlegen
- Spielfeld könnte man auch als Feld betrachten, das wiederum 3 Felder hält

Mehrdimensionale Felder

Hochschule Fulda
University of Applied Sciences

- Für jede weitere Feld-Dimension gibt es eigenes Klammernpaar
- Realisierung des Spielfelds mit zweidimensionalem Feld
  - Z.B.: `char spielfeld[3][3];`
  - Wird Feld direkt mit Mengenschreibweise initialisiert, muss „äußerste“ Dimension nicht angegeben werden, da vom C-Compiler berechenbar

```
char spielfeld[][3] = {
   { ' ', ' ', ' ' },
   { ' ', ' ', ' ' },
   { ' ', ' ', ' ' }
};  //Init. Elem. mit Leerzeichen
```



Zweidimensionales Feld

Hochschule Fulda
University of Applied Sciences

- Wie könnte Spieler *O* gewinnen?

```
spielfeld[0][2] = 'O';
```

  - Zuerst wird Zeile angegeben, später die Spalte
    - Feld wird zeilenweise linear im Speicher abgespeichert
- Ausgabe des Spielfeldes mit verschachtelter Schleife

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        printf(" %c ", spielfeld[i][j]);
    }
    printf("\n");
}
```

	O	
O	X	X

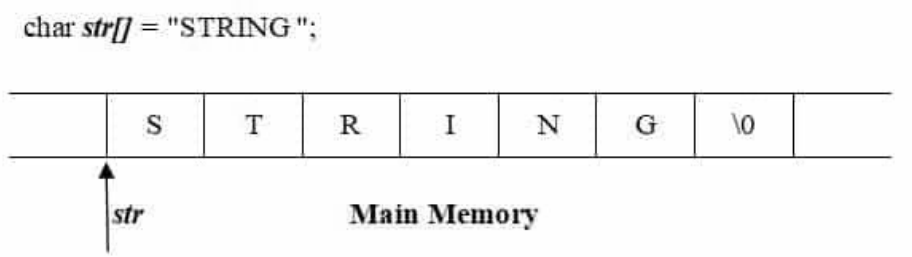
		O
	O	
O	X	X

Grobentwurf Tic Tac Toe

Hochschule Fulda
University of Applied Sciences

- Spielfeld: zweidimensionales Feld
  - Drei mögliche Werte pro Element: `' '`, `'X'`, `'O'`
- Boolesche Variable „fertig“ für Spielende
  - Spielende, wenn kein freies Feld mehr vorhanden oder Spiel gewonnen
- Eingabe des Zugs: Setzen des Steins
- Prüfen, ob dies ein gültiger Zug ist
  - Falls nein, neue Eingabe
- Prüfen, ob Spiel gewonnen
  - Falls ja, Spiel zu Ende
  - Falls nein: Spielerwechsel

Zeichenketten



Zeichenkette (String)

Hochschule Fulda
University of Applied Sciences

- In C kein eigener Datentyp, da nur Feld von Zeichen (Character-Array)
- Lässt sich aber bequem als sog. String-Literal definieren
  - Bsp.: `char str[] = "Hello World";`
  - Terminiert mit String-Endezeichen '\0' (wird bei Literal automatisch angefügt)
- Bei Vereinbarung wird Speicherplatz passender Länge reserviert
- Belegt (wie Array auch) Speicherblock (d.h. Sequenz von Adressen)
  - Bezeichner (hier im Bsp. *str*) ist von Natur aus ein (konstanter) Zeiger
- Kann mit printf()/scanf() ausgegeben/eingelesen werden (mittels **%s**)
  - Variable wird dabei kein **&** vorangestellt, da Variablenwert schon Adresse

C-Strings sind Felder

Hochschule Fulda
University of Applied Sciences

- String in C ist Array von Characters
  - Bsp.: `char str[] = "Hello World";`
  - Bzw.: `char str[] = {'H','e','l','l','o',' ','W','o','r','l','d','\0'};`
    - Beides gleichbedeutend, Strings immer terminiert mit Character `'\0'`
    - Feld besteht im Bsp. aus insges. 12 Zeichen, die Länge des Strings selbst ist aber nur 11
- Bearbeitung und Ausgabe von Strings

```
str[0] = 'H';
str[1] = 'i';
printf("%s\n", str);      Ausgabe: Hillo World

str[2] = '\0';
printf("%s\n", str);      Ausgabe: Hi
```

Strings modifizieren

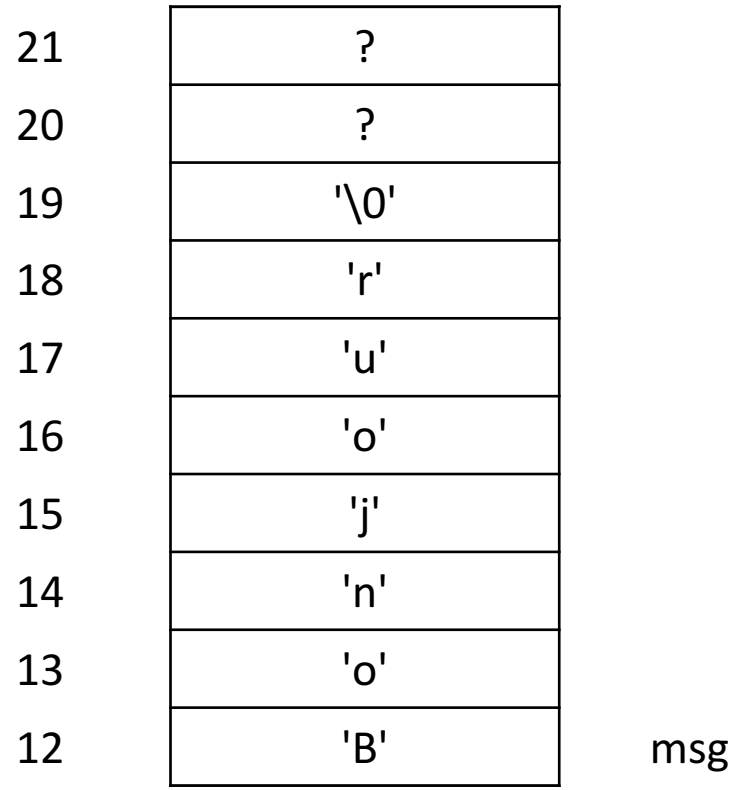
Hochschule Fulda
University of Applied Sciences

- Einlesen von Zeichenketten

```
int tag, jahr;
char monat[20];
// Kein &, Feldname ist schon Adresse
scanf("%d %s %d", &tag, monat, &jahr);
```

- Zuweisung von String-Literal nur bei Initialisierung einer Variablen möglich
- Wie sieht hier der Speicherinhalt aus?

```
char msg[10] = "Bonjour";
msg[0] = 'H';
msg[1] = 'i';
msg[2] = '\0';
```



Strings kopieren

Hochschule Fulda
University of Applied Sciences

- Seien **`s`** und **`t`** Stringvariablen
  - Dann kann man Zeichenkette `t` nicht zuweisen bzw. kopieren mit: `s = t;`
  - Grund: `s` und `t` sind Felder, welche in C unveränderbar die Anfangsadresse des Feldes beinhalten, d.h. Felder sind konstante Zeiger (→ später)
    - Anm.: Wären `s` und `t` "echte" Zeiger, würde nur Zeiger `s` auf Zeiger `t` gesetzt werden
- Implementierungsvorschlag

```
int i = 0;
// Klammerung bei Zuweisung wichtig
while ((s[i] = t[i]) != '\0')
   i++;
```

  - Achtung: hier wird nirgends überprüft, ob Feldlänge überschritten wird ☹

String-Operationen

Hochschule Fulda
University of Applied Sciences

- C kennt keine speziellen Operationen auf ganzen Zeichenketten
  - Es gibt nur Operationen auf einzelnen Zeichen (bzw. auf Feld von Zeichen)
- Strings werden meist mit Hilfe von Bibliotheksfunktionen bearbeitet
  - Dazu erst Header einbinden: `#include <string.h>`
  - Beinhaltet zahlreiche Funktionen auf Strings (z.B. kopieren und konkatenieren)
    - Z.B. *strcpy(dest, src)* oder *strcat(dest, src)* – letzteres hängt Zeichenkette aus *src* an *dest* an
- Bsp.: Strings kopieren mit *strcpy()*

```
char str1[20], str2[] = "Teststring";
strcpy(str1, str2);   // kopiert str2 in str1 (inklusive '\0')
```

  - Achtung: hier wird nicht überprüft, wie viele Zeichen in Zielstring kopiert werden
  - Zugriff auf undefinierten Speicherbereich außerhalb der Feldgrenzen möglich

String-Operationen

Hochschule Fulda
University of Applied Sciences

- Funktion `strcmp(s, t)` vergleicht Zeichenkette *s* mit Zeichenkette *t*
  - Liefert < 0, wenn *s* kleiner ist, 0, wenn beide Strings gleich sind, sonst > 0

    strcmp("A", "A") ist 0

    strcmp("A", "B") ist -1

    strcmp("B", "A") ist 1
    - Vorsicht: strcmp("Z", "a") liefert -1, da ASCII-Werte verglichen werden
- Stringlänge ermitteln mit `strlen(str)`
  - Liefert Länge von *str* (ohne '\0')
  - Bsp.: Welchen Wert liefert strlen(str) hier: 5, 6 oder 8?

    char str[8] = "Hello";
    - Aufruf von strlen(str) liefert 5

Hochschule Fulda
University of Applied Sciences

Anhang

Zufallszahlen

Pseudo-Zufallszahlen

Hochschule Fulda
University of Applied Sciences

```
#include <stdlib.h>
```

- Funktionsprototypen in Standard Library:

```
int rand();
void srand(unsigned int seed);
```

- **rand()** liefert ganzzahlige Zufallszahl zwischen 0 und RAND_MAX
- **srand()** dient zur einmaligen Initialisierung des Zufallszahlengenerators
  - Erzeugte Zahlenfolge hängt vom initialen Wert von Parameter „seed“ ab und wiederholt sich nach bestimmter Anzahl von Aufrufen

Pseudo-Zufallszahlen

Hochschule Fulda
University of Applied Sciences

- Bsp. Würfeln

```
printf("%d ", rand() % 6 + 1);
```

- Problem
  - Jeder Aufruf liefert gleiche Zufallszahlenfolge

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

- Lösung
  - Startbedingungen müssen verändert werden

Pseudo-Zufallszahlen

Hochschule Fulda
University of Applied Sciences

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void) {
  srand(time(NULL));

  for (int i=0; i<20; i++)
    printf("%d ", rand() % 6 + 1);  // Wuerfeln
  printf("\n");
}
```

Timestamp zur Initialisierung des Zufallsgenerators führt dazu, dass erzeugte Zahlenfolge "zufällig" wird

Vielen Dank!

Noch Fragen?