Programmierung 1

Zeiger

Dr. Monika Schak

Woche 8

10. Dezember 2025

Wiederholung - Arrays

- In Arrays kann man Werte desselben Datentyps abspeichern.
- Arrays haben eine feste Länge, die einzelnen Elemente sind aber veränderbar.
- Deklaration von Arrays:
  `int werte[5];` ohne Initialisierung.
  `int werte[] = {1, 2, 3, 4, 5};` mit Initialisierung.
- Eigene Länge ist dem Array nicht bekannt, muss also zusätzlich abgespeichert werden.
- Elementzugriff: `werte[0]`
- Überschreiten der Arraygrenzen wird nicht geprüft, führt aber zu Fehlern!

Wiederholung - Strings

- Strings in C: Arrays bestehend aus `char`s.
- Initialisierung über String-Literal möglich: `char str[i] = "Test";`
- Strings müssen immer mit `'\0'` terminiert werden!
- Beim Einlesen kein Adressoperator nötig:
  `scanf("%s", str);`
- Einige String-Operationen bereits in `string.h`-Bibliothek verfügbar, z.B. kopieren, vergleichen, Länge ermitteln, etc.

Pointer bzw. Zeiger

- Können Speicheradressen von Variablen eines bestimmten Typs speichern.
- Soll z.B. die Adresse einer Integer-Variablen gespeichert werden, ist ein Zeiger vom Typ `int*` nötig, für Float-Variablen ist es analog `float*`.
  Beispiel: `int n = 42, *nP = &n;`
- Steht `*` bei der Variablendeklaration vor dem Variablennamen, handelt es sich um die Deklaration eines Zeigers auf diesen Datentyp.
- Der Adressoperator `&` liefert die numerische Speicheradresse einer Variablen.
- Der Inhaltsoperator `*` liefert den Speicherinhalt der Adresse, auf die der Pointer zeigt. Das Zugreifen auf Daten an dieser Speicheradresse heißt **Dereferenzieren**
  Beispiel: `printf("Wert von n: %d\n", *nP); // 42`

Wo steht was im Speicher?

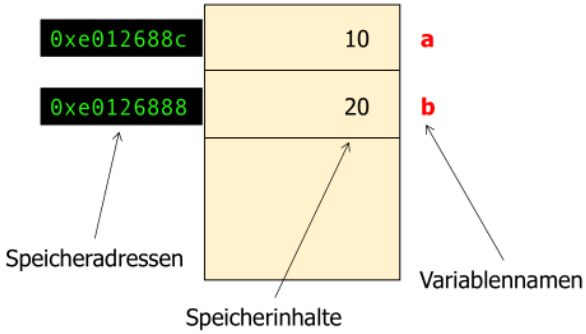
```
#include <stdio.h>

int main () {
  int a = 10;
  int b = 20;

  // some code

  return 0;
}
```

Wie sieht es dann im Hauptspeicher aus?



Wo steht was im Speicher?

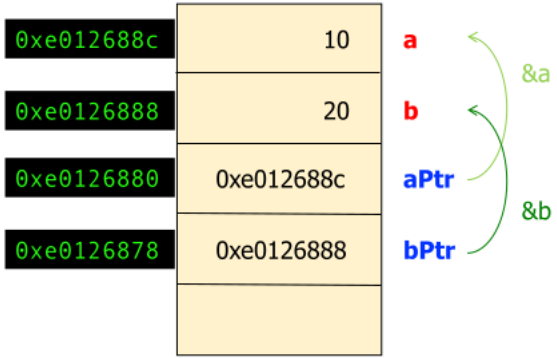
```
#include <stdio.h>

int main () {
  int a = 10;
  int b = 20;

  int *aPtr = &a;
  int *bPtr = &b;
  // some code

  return 0;
}
```

Wie sieht es nun im Hauptspeicher aus?



Was wird ausgegeben?

```
int a = 10, b = 20;
int *aPtr = &a, *bPtr = &b;

printf("%d, %d\n", *aPtr, *bPtr);
printf("%p, %p\n", aPtr, bPtr);
printf("%p, %p\n", &aPtr, &bPtr);
```

Zugriff auf Variable mittels Pointer

```
int i;
int *ip;

i = 5;
ip = &i;
*ip = *ip + *ip;
```

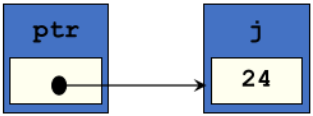
Übung

Welcher Wert wird ausgegeben?

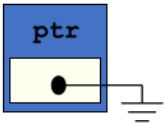
```
int n = 7;
int *y = &n;
int x = *y * 2;
printf("%d\n", x);
```

```
int sum, var = 11;
int *varptr = &var;
*varptr += 4;
sum = var + *varptr;
printf("%d\n", sum);
```

Null-Pointer




```
12
24
0x7b03a928
```




```
Segmentation fault
```

```c
int main()
{
  int j = 12;
  int *ptr = &j;
  printf("%d\n", *ptr);
  j = 24;
  printf("%d\n", *ptr);
  printf("%p\n",  ptr);
  ptr = NULL;
  printf("%d\n", *ptr);  //crash
}
```

Kommandozeile

- Hauptprogramm kann auch Argumente haben:
  `int main (int argc, char *argv[])`
  argument **c**ount: Anzahl der Kommandozeilen-Argumente eines Programms
  argument **vector**: Feld von Zeigern auf Strings, welche die Argumente enthalten
- Bei Aufruf von Konsole aus kann das Programm mit Parametern versorgt werden
  - Ermöglich so z.B. Batch-Verarbeitung
  - Zugriff auf erstes Argument mit `argv[1]`
  - Feld `argv` hat feste Größe `argc`, aber Strings können beliebig groß sein
  - Programmname selbst steht in `argv[0]`

Untypisierte Zeiger: void*

- Zeiger auf `void` bezeichnet Zeiger, der jeden anderen Zeiger ersetzen kann, ohne, dass Information verloren geht
- Jeder Zeiger kann in `void*` verwandelt werden
- Dient als Platzhalter für beliebige Zeiger

Beispiel (mit Type Cast):

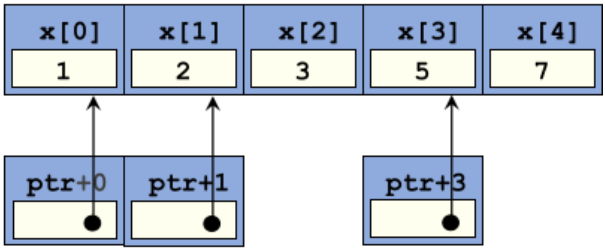
```
int a = 11;
char *cA = NULL;
void *ptr;

ptr = (void*) &a;
ptr = (void*) cA;
```

Pointer und Arrays

- Arrays in C sind konstante Zeiger auf zusammenhängende Speicherblöcke
- Arrayname ohne Index repräsentiert Anfangsadresse des Arrays, ist somit unveränderlicher (konstanter) Zeiger auf erstes Arrayelement
- Jede Operation, die man durch Indizierung von Arrayelementen ausdrücken kann, lässt sich auch mit Zeigern realisieren

```
int main()
{
  int x[5];
  x[2] = 3;
  x[4] = 7;
  int *ptr = x;
  *(ptr+0) = 1;   // x[0] = 1
  *(ptr+1) = 2;   // x[1] = 2
    ptr[3] = 5;   // x[3] = 5
}
```



Zeigerarithmetik

- Ist Rechnen mit Zeigern und Adressen
- Nur Addition bzw. Subtraktion mit Integerwerten ist erlaubt (z.B. `p++`)
  - Zeiger `p` zeigt damit auf nächstes Speicherobjekt des geg. Datentyps (entspricht `p+1`)
  - Achtung: Arrays sind Adresskonstanten, daher bei Arrays Änderung der Anfangsadresse nicht möglich (also z.B. kein Inkrement/Dekrement)
- Ergebnis typabhängig: `p++` bedeutet, dass Größe eines Speicherobjektes des gegebenen Typs auf die Adresse addiert wird
  - `n` Elemente entsprechen `n * sizeof(<Datentyp>)` Byte
  - Ausdruck `p+n` bedeutet also, dass `n`-mal die Datentypgröße auf die in `p` gespeicherte Adresse addiert wird (analog für `p-n`)
  - Damit bezeichnet `p+n` das `n`-te Objekt nach dem Speicherbjekt, auf das `p` gerade zeigt

Zweidimensionale Felder

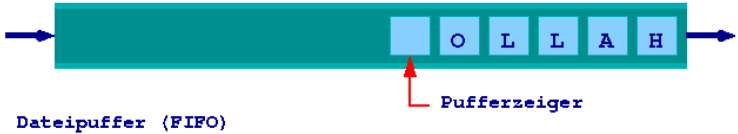
- Zweidimensionale Arrays sind auch über eindimensionale Arrays darstellbar
- `int a[6][10];` reserviert den gleichen Speicher wie `int a[6*10];`
- Zugriff auf Zeile `j` und Spalte `k` ist demnach möglich mit: `a[10 * j + k]` (10 = Anzahl Spalten)

Datenverarbeitung

- Softwaresysteme sind meist auf Ein-/Ausgabe von Daten angewiesen
- Programme sollen nichts „vergessen“, sondern beim nächsten Start mit den Daten persistent weiterarbeiten können
- Je nach Anwendung unterschiedliche Dateiformate gebräuchlich, z.B. Konfigurationsdateien, Tabellen im CSV-Format, Bilddateien (PNG, JPG, ...)
- Dateizugriff ist betriebssystemabhängig
  - Dabei können diverse Probleme auftreten, z.B. Datei wurde nicht gefunden, keine Zugriffsrechte, Festplatte voll, usw.
  - Dateien sind Systemressourcen, daher sollten sie nach der Verwendung unbedingt geschlossen werden. Sonst sind Probleme möglich, z.B. kann eine Datei nicht zum Schreiben geöffnet werden, wenn sie noch zum Lesen geöffnet ist, oder das Betriebssystem-Limit an gleichzeitig geöffneten Dateien ist erreicht

Dateizugriff in C

- Zugrundeliegendes Modell: Datenstrom (Stream)
  - Bytes werden sequentiell gepuffert gelesen oder geschrieben
  - Beispiel: `printf("HALLO WELT\n");`



  - Konzept schon bekannt vom Einlesen eines Zeichens via `scanf("%c");`
- Grundlegender Datentyp: `FILE`
- Vordefinierte Variablen
  - Tastatur: `stdin`
  - Bildschirm: `stdout`, `stderr`

Dateien öffnen und schließen

- Dateien öffnen und schließen kann man über einen sogenannten File Pointer
  Zunächst Variable vereinbaren und initialiseren: `FILE *fp = NULL;`
- Dann Datei öffnen (hier Textdatei)
  Öffnen der Datei *file.txt*: `fp = fopen("file.txt", mode);` Rückgabewert bei Fehlern (z.B. Datei existiert nicht oder falscher Pfad): `NULL`
- Zugriffsmodi (mögliche Werte für mode):
  - `"r"`: zum Lesen öffnen (read), Beginn am Dateianfang, Datei muss existieren
  - `"w"`: zum Schreiben öffnen (write), Beginn am Dateianfang, erzeugt Datei bzw. überschreibt Datei falls diese schon vorhanden war
  - `"a"`: zum Schreiben öffnen (append): Anhängen neuer Daten am Dateiende
- Wichtig: Am Ende alle (vom OS angeforderten) Ressourcen wieder freigeben
  Nach Dateiverarbeitung erfolgreich geöffnete Datei wieder schließen: `fclose(fp);`

Dateien lesen und schreiben

- Formatierte Dateiausgabe über `fprintf()`
  Beispiel (zwei Werte in Datei schreiben, falls Datei erfolgreich geöffnet wurde):

```
if (fp != NULL) {
  fprintf(fp, "%d %d\n", 23, 42);
}
```

- Formatiertes Einlesen von Werten (solange Dateiende noch nicht erreicht):

```
while (!feof(fp)) {
  int a, b;
  // Rueckgabe von [f]scanf: Anzahl korrekt eingelesener Vars
  int i = fscanf(fp, "%d %d", &a, &b);
  if (i < 2) {
    break;
  }
}
```

Zeichenweise Verarbeitung

- Zeichen von Tastatur einlesen (beide Varianten sind identisch)
  - Variante 1: `char c = getchar();`
  - Variante 2: `char c = fgetc(stdin);`
  - Ersetzt man bei `fgetc()` Standardeingabe `stdin` durch einen Dateizeiger auf eine zum Lesen geöffnete Datei, kann man aus der Datei lesen
  - Hat eingelesenes Zeichen den Wert `EOF`, dann wurde das Dateiende erreicht
- Zeichen auf Konsole schreiben (beide Varianten sind identisch)
  - Variante 1: `putchar(c);`
  - Variante 2: `fputc(c, stdout);`
  - Ersetzt man bei `fputc()` Standardausgabe `stdout` durch einen Dateizeiger auf eine zum Schreiben geöffnete Datei, kann man in die Datei schreiben