Zeiger

Proseminar Effiziente Programmierung in C

Johannes Bräuning

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

31.05.2021

Motivation

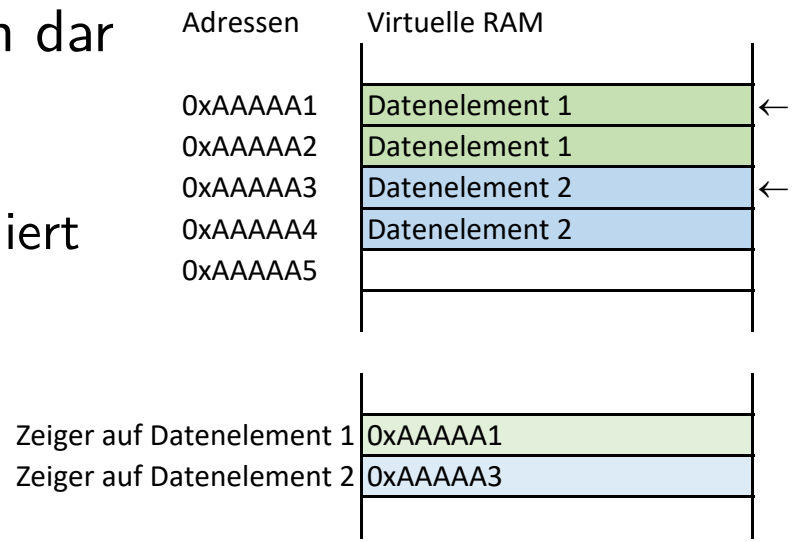
Warum Zeiger verwenden?

- Zugriff auf Daten von anderen Code-Stellen
- Portierbarkeit dynamisch strukturierter Daten bewahren

Gliederung

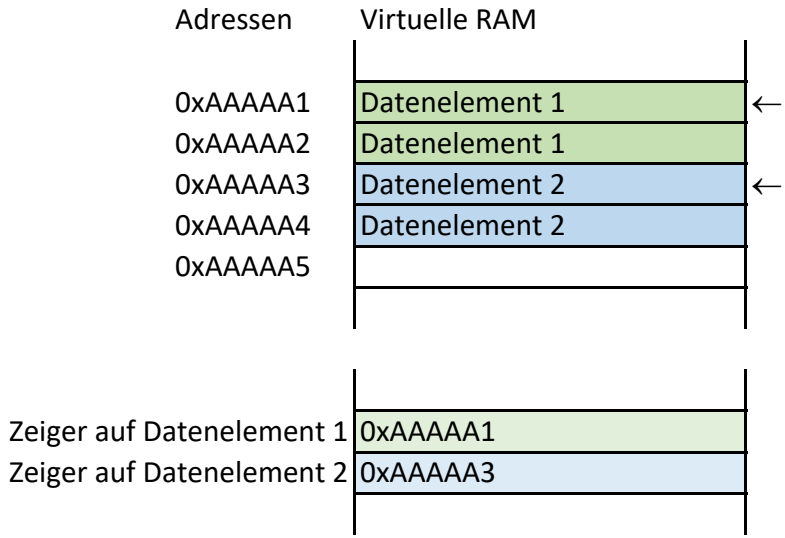
Definition: Zeiger

- stellen virtuelle Speicheradressen dar
- Zeiger sind Variablen
  - werden mit Typ deklariert
  - werden i.d.R. dynamisch definiert



Definition: Operatoren

- address-of Operator: &
  - Adresse von Datenelement
- object-of Operator: *
  - Zeiger auf diesen Typ




```
1 int i = 13;
2 int* a; //a deklariert als Zeiger auf Typ Integer
3 *a = &i; //Zeiger definiert als Adresse von i
4 printf("%d", *a); //13
5 printf("%d", a); //dynamisch zugewiesene Adresse
```

Listing 1: Deklaration und Definition

Eigenschaften: Zustände

- gültig
- null (0)
  - Zeiger definiert mit 0
  - boolescher Wert false
- unbestimmt
  - Zeiger nicht definiert
  - undefiniertes Verhalten
  - nachträglich nicht feststellbar
    $\Rightarrow$ spät deklarieren
    $\Rightarrow$ früh definieren, ggf. zuerst mit 0

Eigenschaften: Ziele I

- Wertvariablen
- Datenstrukturen
  - Operator direkter Zugriff: ->

```
1 //gegeben sei Datenstruktur struktur s mit Feld int f
2
3 struktur* p = &s; //Zeiger p auf s
4
5 printf("%d", p->f); //beide Printaufrufe aequivalent
6 printf("%d", *p.f);
```

Listing 2: Direkter Zugriff auf Datenstruktur

Eigenschaften: Ziele II

- Wertvariablen
- Datenstrukturen
- Felder (engl. arrays)
  - äquivalent zu direktem Zugriff
  - Zeigerarithmetik

```
1 int A[] = {1,2,3,};
2 for(size_t i = 0; i<3; i++) { //Groesse muss bekannt
      ↪ sein
3     printf("%d", *(A+i)); //123
4 }
5 for(size_t i = 0; i<3; i++) {
6     printf("%d", A[i]); //123
7 }
```

Listing 3: Direkter Zugriff auf Feld

Exkurs: Mehrdimensionale Felder

- wie auch z.B. Java
- Zugriff verändert nur innere Argumente
- restliche Argumente nur „Koordinaten“ im Feld
  $\Rightarrow$ verkettete Zeiger

```
1 int A[2][2];
2 A[1][1] = 13;
3 printf("%d", (*(A+1))[1]); //13
```

Listing 4: Mehrdimensionales Feld

komplexeres Listing hierzu: „Modern C“, Jens Gustedt S.117 [1]

Zerfall

- Alle per typedef erzeugten Datentypen...
- Alle Felder mit zur Laufzeit festgelegter Länge...

$\Rightarrow$ Sind eigentlich nur Zeiger ohne weitere Informationen

$\Rightarrow$ Weitergegeben werden also nur eine Adresse und ein Typ!

Zerfall und Portabilität

Motivation: Zeiger können Portierbarkeit dynamisch strukturierter Daten bewahren:

$\Rightarrow$ Typen unabhängig von Implementierung referenzierbar

NULL/ null (0) Zeiger

- mit NULL initialisierter Zeiger: NULL-Zeiger
  - ein Typ
  - aber nicht im C-Standard definiert [2]
    $\Rightarrow$ nicht einheitlich implementiert
    $\Rightarrow$ keine gute Portabilität
- mit 0 initialisierter Zeiger: null-Zeiger
  - je nach Deklaration: verschiedene Typen

Zusammenfassung

- Zeiger: Speicheradresse mit zugehörigem Typ
- Zustände:
  - gültig
  - null
  - unbestimmt
- Zugriffsziele:
  - Wertvariablen
  - Datenstrukturen
  - Felder
- Zerfall
- NULL-/ null (0) -Zeiger

Literatur

[1] [Modern C, Jens Gustedt, 2018, URL: https://gforge.inria.fr/frs/download.php/file/37813/ModernC.pdf]

[2] [Cert C Coding Standart, eingesehen 28.05.2021 (noch online 30.05.2021), URL: https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers]

[3] [In C Language How to Use Pointer to Access Two-Dimensional Array Element, WeiQing Bai, 2013, URL: https://www.atlantis-press.com/proceedings/isccca-13/5761]

[4] [cppreference.com]

[5] [Zeiger in C, Wikipedia, eingesehen 09.05.2021 (noch online 30.05.2021), URL: https://de.wikipedia.org/wiki/Zeiger_in_C]