Tut 5 (Tutor)

Friday, 17. November 2017 08:42

0. ORGANISATORISCHES

- 1. Block-Test bis So. 23^{59}

1. WIERHOLUNG

#throwback. Was kam letzte Woche?

- Arrays
- Pointer

- Codebeispiel 1:
  - → Call-by-reference?
    - Es wird die Adresse übergeben, nicht eine Kopie (vgl.: Call-by-Value)
    - swap(&x, &y) vertauscht Var. direkt in main()
- Codebeispiel 2:
  - → Array übergeben?
    - Nur die Referenz!
    - source wird gespiegelt in target geschrieben
    - Fangen bei 0 an zu zählen! (i-1)

3. REKURSION

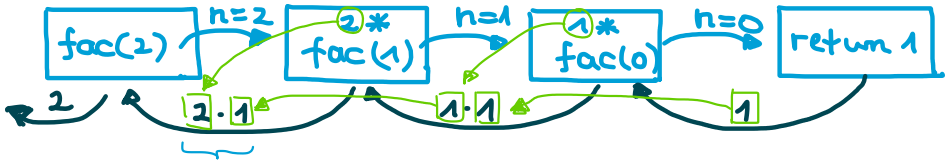
3.1 Rekursive Funktionen

- WTF?
  - → Funktionen, die sich selbst aufrufen
- Bsp.: unendlich rekursiv?
  - → Benötigt Abbruchbedingung, nicht rekursiv

REKURSIVE FUNKT.

... Fkt. die sich sebst aufrufen

Bsp.: Fakultät. Was passiert bei fac(2)?



C, CL

2! = 2 → Ergebnis

- Jede lineare Rekursion kann iterativ umgeschrieben werden
  - ↳ also mit Schleifen
- Aufg. 12 a) durchsprechen
- Geht auch umgekehrt
- Aufg. 12 b) durchsprechen
  - Werde Zeichenumwandlung rekursiv an

- jedes mal str+1 bis '\0' erreicht

- Iterationen bevorzugt. Warum?
  - Weniger Zwischenergebnisse
  → Bsp. Fakultät: f*= → immer in f gespeichert
  - weniger Speicherverbrauch:
  → Funktionsaufrufe beanspruchen Stackspeicher:
  ↳ nicht nur Variablenspeicher, auch Rücksprungadresse

2. DYNAMISCHE SPEICHERVERWALTUNG

1. Einführung

→ Folien unter www.thevfdcollective.com/blog/inftech

- Wie sind Daten im Arbeitsspeicher angeordnet?
  → Letzte Woche: Daten sind im RAM und besitzen eindeutige Adresse!
- Programm 'sieht' 4 GB (virt.) Speicher
- Der 'uninteressante' Teil: 'Unten' befindet sich der Programmcode, dann: globale Variablen & Konstanten
- ① Der Stackspeicher.
  → Wenn eine Funktion aufgerufen wird, wächst der Stackspeicher (oben → unten)
  → So viel Platz wie benötigt für lokale Variablen (und Fkt.-Parameter), alles in den Stack
  → Wird Fkt. fertig ausgeführt, wird der beanspruchte Stack freigeräumt
  ↳ Variablen werden wieder freigegeben
- Warum funktioniert noheap.c nicht?
  - Normale Definition erzeugt Array im Stack
  - destination wird beim Verlassen gelöscht!
- Wie z.B. Array über mehrere Scopes erhalten?
  → Speicherfreigabe verhindern (Folie)
  → Antwort: Statt Stackspeicher einfach Heap-Speicher verwenden!

2.2 Dynamische Speicherverwaltung

- Dynamische Daten auf dem Heap bestehen so lange, bis sie manuell, → explizit freigegeben werden!
- Programmierer verwaltet!
- Wie sieht der Syntax hierfür aus?

> Dyn. Speicherverw.
> z.B 200 ints auf dem Heap
> `int * arr = malloc(200* sizeof(int));`
> ↓

Wie viele Bytes ?
200 * 4 Bytes = 800 Bytes

```
int* arr = calloc(200, sizeof(int));
```
↳ alternativ, Init. mit 0

C, CL

- realloc erwähnen
- Speicher muss auch wieder freigegeben werden
  → Ansonsten entstehen Speicherlecks → BÖSE !
  → Lecks belegen Speicherplatz und verlangsamen Programm

Speicher freigeben: free !
Bsp.: `free(arr);`

C, CL

- Was passiert bei zu wenig Speicher ?
  → Abfrage bei Allokation !
- noheap.c mit Heap ?

4. STRUKTUREN

4.1 Anlegen

- „Objektorientierung" in C
- Wozu ? Mehrere Variablen zu einem Datentyp
  → Sinnvoll auch, um mehrere Var. zu returnen
- Wie implementieren wir eine Struktur ? (typedef)

STRUKTUREN

Bsp. Komplexe Zahl

```
struct complex {
    double re;
    double im;
};
```
← wichtig

C, CL

4.2 Strukturen auf dem Stack

- Instanz der Struct erzeugen ?

```
struct complex z1;
struct complex z2 = {1, -1};
z1.re = 2;
z1.im = 2;
struct complex z3 = {z1.re + z2.re,
                     z1.im + z2.im};
```

C, CL

4.3 Strukturen auf dem Heap

- Complex-Codebeispiel
- Hardcore-Codebeispiel: Struct-Array
  - Stack - Stack
  - Arr Heap, Struct stack v.v.
  - Heap, heap

5. Quiz

- Quiz C