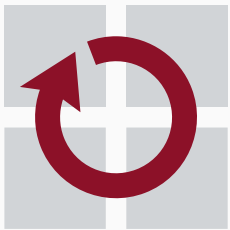
Übungen zu Systemprogrammierung 1

Ü6 – Besprechung A3 & Testen

Sommersemester 2025

Luis Gerhorst, Thomas Preisner, Tobias Häberlein, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Agenda

13.1 Besprechung Aufgabe 3: halde

13.2 Datentypen nach C11

13.3 Zeiger und undefiniertes Verhalten

Aufgabe 3: halde

Aufgabenstellung

- Eigene **Freispeicher**verwaltung
- `malloc()`, `realloc()`, `calloc()` & `free()`
- Makefile
- Testfälle

Vorstellen der Lösung von...

Standard Datentypen

- Größe der Datentypen ist von der Ausführungsumgebung abhängig
  $\Rightarrow$Hardware oder Betriebssystem
- Datentypen können mit `unsigned` als vorzeichenlos deklariert werden
- Typen und vorgegebene Wortbreiten [1] in Bytes:

Typ	relative Größe	C11	64-Bit Linux
`char`		1	1
`short int`	$\geq$ `char`	≥ 2	2
`int`	$\geq$ `short`	≥ 2	4
`long int`	$\geq$ `int`	≥ 4	8
`long long int`	$\geq$ `long int`	≥ 8	8

[1] Eigentlich ist der Wertebereich und nicht die Wortbreite definiert

Exakte Wortbreiten

- exact-width integer geben Garantie über die tatsächliche Wortbreiten
- Die Datentypen lauten: `{u}int{8,16,32,64}_t`
- Limit-Makros definiert in `stdint.h`:
  `UINT{8,16,32,64}_MAX`
  `INT{8,16,32,64}_{MAX,MIN}`
- Makros für Format-Strings [2] (siehe `inttypes.h(7p)`)

```
uint8_t  c = 0x11;
uint16_t s = 0x1122;
uint32_t i = 0x11223344;
uint64_t l = 0x1122334455667788;
printf("%"PRIu8"\n", c);
printf("%"PRIu16"\n", s);
printf("%"PRIu32"\n", i);
printf("%"PRIu64"\n", l);
```

```
int8_t  c = -128;
int16_t s = -32768;
int32_t i = -2147483648;
int64_t l = -9223372036854775808;
printf("%"PRIi8"\n", c);
printf("%"PRIi16"\n", s);
printf("%"PRIi32"\n", i);
printf("%"PRIi64"\n", l);
```

[2] Konkatenation von String Literalen, wenn sie direkt aufeinander folgen.
`char *x = "Hallo" "Welt";` $\iff$ `char *x = "HalloWelt";`

Weitere Ganzzahl-Typen

{s}size_t

- `size_t` ist der Ergebnis-Typ des `sizeof`-Operators
- Beschreibt die Größe von Datentypen in Byte
- Das Makro `SIZE_MAX` beinhaltet den maximal Wert
- `ssize_t` ist vorzeichenbehaftet und wurde mit POSIX eingeführt
- Der max. Wert ist `SSIZE_MAX`
- Formatstring mit “%uz“ bzw. “%z“ für `ssize_t`

ptrdiff_t

- Vorzeichenbehafteter Ganzahl-Typ, der die Differenz von zwei Pointern speichern kann
  - Formatstring mit ”%td”

Weitere Ganzzahl-Typen II

```c
#include <stdio.h>
#include <stddef.h>

#define SIZE 10

int main(void) {
  int i_arr[SIZE] = {1};
  char *start = (char *) &i_arr[0];
  char *end = (char *) &i_arr[SIZE];

  ptrdiff_t b_diff = end - start;
  printf("Byte diff: %td\n", b_diff);
}
```

```
> gcc -o ptrdiff ptrdiff.c && ./ptrdiff
byte diff: 40
```

Weitere Ganzzahl-Typen II

- Fehler und Ergebnis werden mit dem selben Mechanismus mitgeteilt
- Zur Unterscheidung wird der Wertebereich des Ergebnis aufgeteilt
- `void *malloc(size_t size);`
  - Nutzt den Wert `NULL` als Rückgabewert im Fehlerfall
  - Alle anderen Werte sind gültige Zeiger auf allokierten Speicher
- `ssize_t write(int fd, const void *b, size_t nbytes)`
  - `ssize_t` ist das vorzeichenbehaftete Pendant zu `size_t`
  - Der Rückgabewert beschreibt die Anzahl der geschriebenen Bytes
  - Negativer Rückgabewerte beschreibt Fehler
  - $\Rightarrow$ `nbytes` darf nicht größer als `SSIZE_MAX` sein
  - $\Rightarrow$ Der Wertebereich halbiert sich

Überläufe

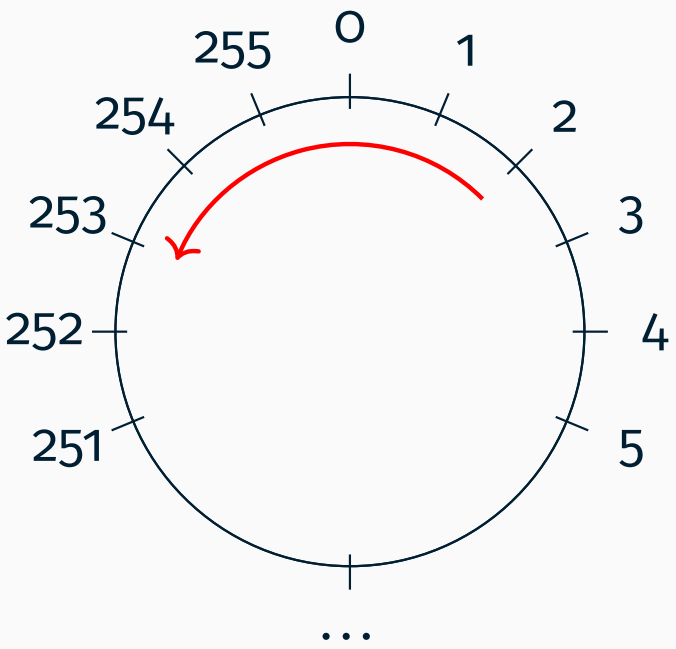
```c
#include <stdio.h>
#include <limits.h>

int main(void) {
  int i = INT_MAX + 1;
  unsigned u = UINT_MAX + 1;
  printf("%d %u\n", i, u);
}
```

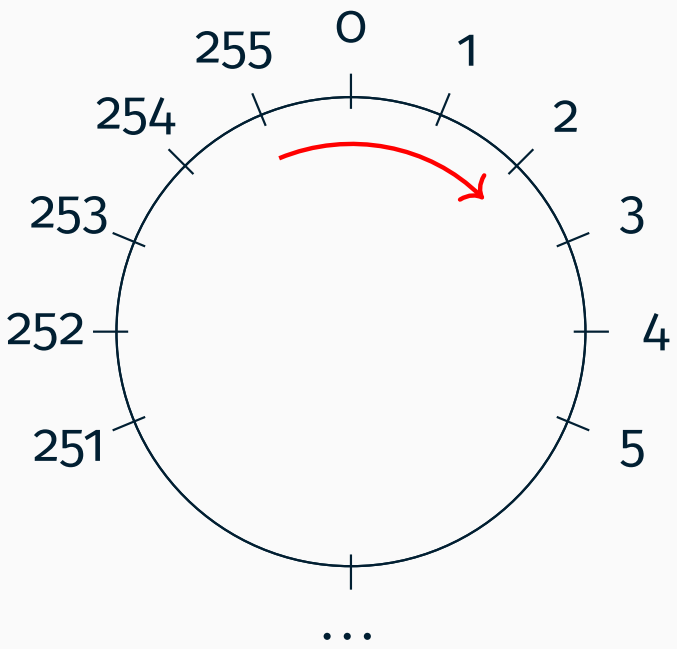
- `gcc -Wall -Werror` warnt vor „overflow“ für `INT_MAX + 1`
- Das Ergebnis ist zu groß für `int` und Überläufe sind bei vorzeichenbehaftete Datentypen undefiniert
  - Meist folgt `INT_MIN` auf `INT_MAX`. Dies ist aber **nicht garantiert**
- Bei vorzeichenlosen Datentypen hingegen, ist der Überlauf definiert
  - Auf `UINT_MAX` folgt die `0`

Überläufe mit Umbruch

```
uint8_t i = 2;
i -= 5;
/* i = 253 */
```




```
uint8_t i = 255;
i += 3;
/* i = 2 */
```



calloc(3p): Umbruch vorzeichenloser Ganzzahlen

calloc(3p) allokiert Speicher für ein Array mit nmemb Elementen, die jeweils size Byte groß sind.

```
static void *calloc(size_t nmemb, size_t size) {
  size_t total = nmemb * size;
  void *p = malloc(total);
  if (p) {memset(p, 0, total);}
  return p;
}
int main(void) {
  size_t s = 16;
  void *p = calloc((SIZE_MAX/16)+3, s); // mehr als SIZE_MAX bytes
  if (p) { memset(p, 0xff, 5 * s); }
}
```

- size_t ist groß genug jeden der Operanden zu speichern
- Das Produkt der Multiplikation ist jedoch zu groß
- total wird der Wert 32 zugewiesen (bei 8 Byte Wortbreite)
- memset in main schreibt über den zugewiesenen Buffer hinaus

calloc(3p): Von den Großen lernen

```
void *calloc(size_t m, size_t n)
{
  if (n && m > SIZE_MAX/n) {
    errno = ENOMEM;
    return 0;
  }
  n *= m;
  /* ... */
}
```

Listing 1: musl src/malloc/-malloc.c (version 1.1.22) leicht abgeändert

- Explizite Prüfung, ob ein Überlauf stattfinden kann
- Stellt sicher, dass nicht durch 0 geteilt wird
- Division ist eine langsame Operation
- Moderne Übersetzer erkennen die Überlaufprüfung und ersetzen sie durch effiziente Befehle

Undefiniertes Verhalten

> *When the compiler encounters [a given undefined construct] it is legal for it to make demons fly out of your nose. (⇝ Jaron File)*

- Nicht alles ist im C-Standard definiert
- Z.B. Vergleiche von Zeigern, die auf versch. Arrays zeigen
- In solchen Fällen darf der Übersetzer „frei gestalten“
- Der übersetzte Code kann der Erwartung des Programmierers widersprechen

Beispiel für undefiniertes Verhalten

```c
#include <stdio.h>
int main(void) {
  long x[4], y[4];
  void *p = &x[0], *q = &y[4];
  printf("%p %p %d\n", p, q, p == q);
}
```

```
> gcc -O0 ./ptr_comparison.c && ./a.out
0x7ffd4c6eae20 0x7ffd4c6eae20 1
> gcc -O2 ./ptr_comparison.c && ./a.out
0x7ffec72d95d0 0x7ffec72d95d0 0
```

- Ohne Optimierung: Vergleich der Zeiger während der Laufzeit
- Mit -O2: Auswertung des Vergleichs p==q während der Übersetzung
- Vergleich ist undefiniert, da p und q auf versch. Objekte zeigen
- Der Übersetzer darf annehmen, dass beide Zeiger verschieden sind[3]

[3] Details im Appendix

Appendix

Details zum undefinierten Zeigervergleich (1)

```c
#include <stdio.h>
int main(void) {
long x[4], y[4];
void *p = &x[0], *q = &y[4];
printf("%p %p %d\n", p, q, p==q);
}
```

```
> gcc -O0 ./ptr_comparison_01.c
> ./a.out
0x7ffd4c6eae20 0x7ffd4c6eae20 1
```

```c
#include <stdio.h>
int main(void) {
long x[4], y[4];
void *p = &x[4], *q = &y[0];
printf("%p %p %d\n", p, q, p==q);
}
```

```
> gcc -O2 ./ptr_comparison_02.c
> ./a.out
0x7ffec72d95d0 0x7ffec72d95d0 0
```

- Verwendet wurde der gcc in der Version „8.3.0 (Debian 8.3.0-6)“ aus dem Debian Package-Repository „buster“
- Der Übersetzer ordnet je nach Optimierungsstufe die Arrays auf dem Stack um