Programmieren in C

SS 2021

Vorlesung 11, Montag 05. Juli 2021

(Funktionszeiger)

Prof. Dr. Peter Thiemann
Lehrstuhl für Programmiersprachen
Institut für Informatik
Universität Freiburg

Teile der Folien enthalten Material von Prof. Dr. Bast

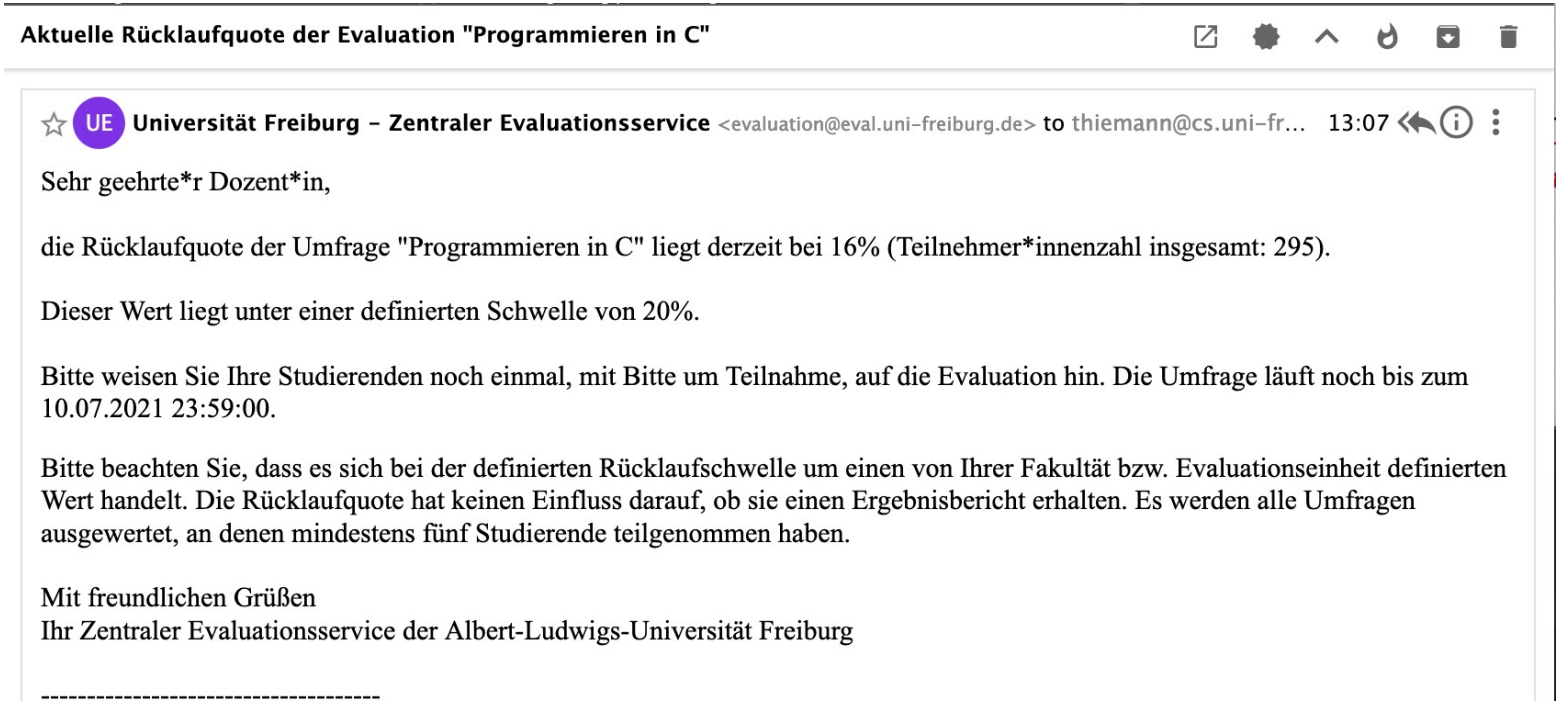
Heutige Vorlesung

UNI FREIBURG

- Organisatorisches
  - Evaluation
  - Erfahrungen mit dem Ü10 — Life
- Inhalt
  - Funktionszeiger — Deklaration, Verwendung
  - (Kommandozeilenoptionen) — getopt_long

Übung: Fun mit Funktionszeigern

Orga 1 - Evaluation



Aktuelle Rücklaufquote der Evaluation "Programmieren in C"

Universität Freiburg – Zentraler Evaluationsservice <evaluation@eval.uni-freiburg.de> to thiemann@cs.uni-fr… 13:07

Sehr geehrte*r Dozent*in,

die Rücklaufquote der Umfrage "Programmieren in C" liegt derzeit bei 16% (Teilnehmer*innenzahl insgesamt: 295).

Dieser Wert liegt unter einer definierten Schwelle von 20%.

Bitte weisen Sie Ihre Studierenden noch einmal, mit Bitte um Teilnahme, auf die Evaluation hin. Die Umfrage läuft noch bis zum 10.07.2021 23:59:00.

Bitte beachten Sie, dass es sich bei der definierten Rücklaufschwelle um einen von Ihrer Fakultät bzw. Evaluationseinheit definierten Wert handelt. Die Rücklaufquote hat keinen Einfluss darauf, ob sie einen Ergebnisbericht erhalten. Es werden alle Umfragen ausgewertet, an denen mindestens fünf Studierende teilgenommen haben.

Mit freundlichen Grüßen
Ihr Zentraler Evaluationsservice der Albert-Ludwigs-Universität Freiburg

---

Orga 2 - Punkte

UNI FREIBURG

Punkte

Name	Minimum	Quartile 1	Median	Quartile 3	Maximum	Average	Mean Deriv	Variance	Samples
blatt01	0.00	16.00	16.00	16.00	16.00	14.07	3.02	21.26	269
blatt02	0.00	11.00	14.00	15.50	16.00	11.80	4.17	28.76	273
blatt03	0.00	6.00	14.00	15.50	16.00	10.58	5.35	37.20	274
blatt04	0.00	3.00	12.00	15.00	16.00	9.36	5.65	38.19	271
blatt05	0.00	0.00	11.50	15.00	16.00	8.88	6.21	44.02	273
blatt06	0.00	0.00	9.00	14.50	16.00	7.96	5.55	37.83	269
blatt07	0.00	0.00	16.00	22.00	22.00	12.68	8.58	87.63	272
blatt08	0.00	0.00	9.00	14.50	16.00	8.23	5.53	38.74	261
blatt09	0.00	0.00	5.00	14.00	16.00	6.56	6.08	42.56	255
Total	0.00	2.00	13.00	15.50	22.00	10.04	6.12	47.12	2417

Zeitbedarf

Name	Minimum	Quartile 1	Median	Quartile 3	Maximum	Average	Mean Deriv	Variance	Samples
Blatt 01	0.33	2.0	3.75	5.0	25.25	4.03	1.89	8.69	235
Blatt 02	0.33	3.0	4.0	6.0	36.0	4.94	2.14	12.04	224
Blatt 03	0.0	4.5	6.0	8.0	24.0	6.44	2.39	10.82	217
Blatt 04	0.5	5.0	6.5	9.0	30.0	7.16	2.68	13.82	211
Blatt 05	0.5	5.5	7.5	9.0	29.0	7.79	2.61	14.79	194
Blatt 06	0.0	6.0	8.5	11.0	48.0	9.11	3.24	22.78	192
Blatt 07	0.0	5.0	6.5	9.5	48.0	7.72	3.02	23.24	184
Blatt 08	0.0	4.0	6.0	8.0	25.0	6.28	2.75	13.76	180
Blatt 09	0.0	4.0	6.78	9.0	36.0	7.19	3.27	22.04	157
Blatt 10	0.0	4.33	6.0	8.0	25.0	6.6	2.59	13.16	146

Orga 3 - Erfahrungen

Zeitaufwand: 13:30

Im Forum gefragt?

<...Andere Erfahrungen...>

Bisher mit Abstand die unschönste Übung. Ich habe keine Idee, wie sich die Regeln für die Überlappung schön implementieren lassen, so wie ich es gemacht habe, kann es aber defintiv NICHT stimmen. Die Übung habe ich abgegebrochen, da ich bereits genug Punkte habe und der Frust exponentiel angestiegen ist. Die nicht-funktionierenden Funktionen habe ich auskommentiert, sodass das es immerhin möglich ist, die Dateien zu kompilieren. Fragwürdig finde ich auch die Punkteverteilung: 2 Punkte für die Teilaufgaben (a) bis (g)? Zugegebenermaßen: manche der Teilaufgaben waren geschenkt, aber in der Summe würde ich ihnen mehr als nur 2 Punkte zusprechen. Ich hoffe, dass die Bepunktung im Abschlussprojekt etwas ausgeglichener wird.

Orga 4 - Erfahrungen

Zeitaufwand:
8:00

Ich fand die Aufgabe sehr gut, und eigentlich sehr spannend. Ich habe davor

auch Game of Life in Python gemacht so die Idee war mir schon bekannt. Die kleine

Freiheiten hatten mir auch gut gefallen. Mit bekannten Code zu arbeiten macht immer

viel mehr Spaß, es gefällt mir wie wir alles auf einenader bauen. Bis load file

ging alles hervorragend. Dann hatte ich große Probleme gehabt. Erst die Address Sanitizer Errors, dann die falsch gedruckte Zeilen..

Ich habe viel Zeit in Stackoverflow verbracht. Ich habe auch wie im Vorlesung

gezeigt die man 3 gelesen. Am Ende hat es endlich geklappt. Ich habe nicht erwartet dass es so schwierig sein würde.

Allgemein alles auf dem Papier zu zeichnen hatte mir viel geholfen. Sonst

im Vorlesung ich hatte mich aufmerksam über diese Schreibweise mit "?" gemacht,

und es freut mich dass ich es in dieser Übung selbst ausprobieren konnte.

Orga 5 – Erfahrungen

UNI FREIBURG

- Ich schäme mich ein bisschen, wieder nichts abzugeben, aber ich habe die Hände voll mit anderen Fächern und hier sind die 50% eben schon drin.
  - Alles gut…
- unerwartet muss ich meine Pause verlaengen. gut, dass es noch ein Blatt gibt. und ich habe mich gefragt, wann game of life als Aufgabe kommtX)
- An sich eine coole Idee, aber das zu debuggen ist die Hölle...
- Aber ich schätze so ist das Spiel des Lebens...

Funktionszeiger 1/7

UNI FREIBURG

- Von jeder Funktion kann mit dem Adressoperator & die Adresse bestimmt werden.
- Der resultierende Funktionszeiger kann als Parameter verwendet werden, als Ergebnis zurückgegeben werden oder in Datenstrukturen abgelegt werden – wie jeder andere Zeiger.
- Über den Funktionszeiger kann die Funktion aufgerufen werden.
- Die Schreibweise für den Typ eines Funktionszeigers ist gewöhnungsbedürftig…

Funktionszeiger 2/7

UNI FREIBURG

- Beispiel 1

```
/* Apply function `func` to every element of `xs` */

void ivec_iter(Ivec *xs, void (*fp) (int*));
```

  - Das Argument fp ist ein Zeiger…
  - Auf eine Funktion, die void zurückliefert, und
  - Einen Zeiger auf int als Argument nimmt!
  - (Intention: über den Zeiger kann das Element im Vektor gelesen und überschrieben werden)

- Beispielhafter Aufruf

```
ivec_iter(xs, &do_abs);
```

Funktionszeiger 3/7

- Beispiel 2

```
/* Apply function `fp` to every element of `xs` with current
argument count */

void ivec_iteri(Ivec *xs, void (*fp) (int, int*));
```

  - Das Argument fp ist ein Zeiger…
  - Auf eine Funktion, die void zurückliefert, und
  - Ein int und einen Zeiger auf int als Argument nimmt!

- Beispielhafter Aufruf

```
ivec_iteri(xs, &printi);  /* print indexed elements of xs */

void printi(int i, int *xi) {

   printf("%5d: %5d\n", i, *xi);

}
```

Funktionszeiger 4/7

UNI FREIBURG

- Beispiel 3 (für den generischen Vektor)

```
/* Apply function `fp` to every element of `xs` */
void vec_forall (Vec *xs, void (*fp) (void*));
```

  - Das Argument fp ist ein Zeiger…
  - Auf eine Funktion, die void zurückliefert, und
  - Einen Zeiger auf void als Argument nimmt! (nur lesend!)

- Beispielhafter Aufruf

```
vec_forall(xs, &json_value_free);  /* free elements of xs */
```

Funktionszeiger 5/7

UNI FREIBURG

- Beispiel 4 (für den generischen Vektor)

```
/* Iterate function `fp` over all elements of ‘xs‘ */

void vec_iter (Vec *xs, void (*fp) (void**));
```

  - Das Argument fp ist ein Zeiger…
  - Auf eine Funktion, die void zurückliefert, und
  - Einen Zeiger auf einen Zeiger auf void als Argument nimmt! (Lesen und Schreiben möglich!)

- Beispielhafter Aufruf

```
vec_iter(xs, &json_value_increment);  /* increment xs */
```

Funktionszeiger 6/7

UNI FREIBURG

- Typdeklaration für Funktionszeiger
- Manchmal unvermeidlich (zB Rückgabetyp einer Funktion)
- Struktur wie Variablendeklaration

```
typedef int (*compar_t)(const void *, const void *);
```

- Definiert den Typ compar_t als
  - Zeiger auf eine Funktion mit
  - Rückgabewert int
  - Zwei Argumenten vom Typ const void*
- Verwendung als Comparator (Vergleichfunktion)
  - Vgl int strcmp(const char* x, const char *y)
- Verwendet in Implementierung einer generischen Sortierfunktion!

Funktionszeiger 7/7

UNI FREIBURG

- man 3 qsort
- man 3 versionsort

```
typedef int (*compar_t)(const void *, const void *);
```

- Anwendung: scandir()

Parsen von Optionen 1/6

UNI FREIBURG

- Beispiel mit "langen" Optionennamen
  - Typischer Aufruf von der Kommandozeile

    ./InputOutputMain --head=3 --numbers example.csv
  - Zur Erinnerung: Argumente der main Funktion

    int main(int argc, char** argv);
  - Die Werte von argv sehen dann so aus:

    argv[0] : "./InputOutputMain"
    argv[1] : "--head=3"
    argv[2]: "--numbers"
    argv[3] : "example.csv"

Parsen von Optionen 2/6

UNI FREIBURG

- Beispiel mit "kurzen" Optionennamen
  - Äquivalenter Aufruf zu dem von der Folie vorher:

    ./InputOutputMain -h 3 -n example.csv
  - Argumente der main Funktion

    int main(int argc, char** argv);
  - Die Werte von argv sehen jetzt so aus:

    argv[0] : "./InputOutputMain"
    argv[1] : "-h"
    argv[2] : "3"
    argv[3]: "-n"
    argv[4] : "example.csv"

Parsen von Optionen 3/6

- Verarbeitung mit getopt, Teil 1/3
  - Im Programm muss ein Feld von Strukturen definiert werden, in dem die Optionen definiert werden (langer Name, Status des Arg., NULL, kurzer Name)

```
#include <getopt.h>

struct option options[] = {
   { "head", required_argument, NULL, 'h' },
   { "numbers", no_argument, NULL, 'n' },

   { NULL, 0, NULL, 0 }              // End of array.
};
```

  - An der dritten Position kann statt NULL ein Zeiger auf eine Variable (Typ int*) stehen

    Siehe "man 3 getopt" für die Semantik davon

Parsen von Optionen 4/6

- Verarbeitung mit getopt, Teil 2/3
  - Verarbeiten der Optionen:

```
optind = 1;                          // Start with argv[1].
while (true) {
   // s and n = short names, the : means with argument.
   char c = getopt_long(argc, argv, ”h:n", options, NULL);
   if (c == -1) break;               // No more options.
   switch (c) {                      // c is the short name.
   case 'h’: head = atoi(optarg);    // Argument in optarg.
             break;
   case 'n': numbers = true;         // Option without arg.
  }
}
```

Parsen von Optionen 5/6

UNI FREIBURG

- Verarbeitung mit getopt, Teil 3/3
  - Jetzt noch die Nicht-Options Argumente

    ```
    if (optind + 1 != argc) { printUsageAndExit(); }
    fileName = argv[optind];
    ```

  - Achtung: optind zeigt nach der vorherigen Schleife auf das nächste Argument, das keine Option mehr ist
  - Das funktioniert, weil die Schleife auf der Folie vorher die Optionen und Ihre Argumente "nach vorne tauscht"

    Vorher: ./InputOutputMain –h 3 example.csv –n
    und optind == 1, so dass argv[optind] == "-h" ist

    Nachher: ./InputOutputMain –h 3 –n example.csv
    und optind == 4, so dass argv[optind] == "example.csv"

Parsen von Optionen 6/6

UNI FREIBURG

- Zwei wichtige globale Variablen aus getopt.h
  - **optind** ist der Index von dem Argument, das getopt_long als nächstes bearbeitet

    optind ist zwar mit 1 initialisiert, aber Achtung:

    beim Testen wird die getopt_long Schleife evtl. mehrmals hintereinander ausgeführt, deswegen vorher immer optind = 1 setzen
  - **optarg** ist das Argument der zuletzt verarbeiteten Option, sofern sie ein Argument hatte (sonst NULL)

    optarg ist immer vom Typ char*; bei Bedarf selbst

    konvertieren

Literatur / Links

UNI FREIBURG

- Parsen von Optionen
  - man 3 getopt