Programmieren in C

Prof. Dr. Peter Thiemann
Hannes Saffrich
Sommersemester 2021

Universität Freiburg
Institut für Informatik

Übungsblatt 4

Abgabe: Montag, 17.05.2021, 9:00 Uhr morgens

In diesem Übungsblatt geht es um Zeiger, dynamisch allozierten Speicher und Datenstrukturen.

Wie im vorherigen Übungsblatt, können Sie bei den Fingerübungen Ihre Definitionen und Tests in eine einzige `.c`-Datei schreiben.

Die Beispiele auf den Übungsblättern dürfen übrigens auch in Ihren Unittests verwendet werden.

Aufgabe 4.1 (Fingerübungen: `structs` und Zeiger; 3 Punkte)

Diese Aufgabe soll in der Datei `warmup_structs.c` bearbeitet werden. Schreiben Sie sinnvolle Unittests für jede Funktion die Sie definieren.

(a) (0.5 Punkte) Definieren Sie zwei Funktionen `add_3` und `add_assign_3`, die sich wie folgt verhalten:

```
int i = 5;
i = add_3(i);

int j = 5;
add_assign_3(&j);

/* i and j are now both 8. */
```

(b) (0.5 Punkte) Definieren Sie ein `struct Point` mit zwei Feldern `x` und `y` vom Typ `int`.

(c) (1 Punkt) Definieren Sie zwei Funktionen `add_point` und `add_assign_point`, die sich wie folgt verhalten:

```
struct Point p1 = {1, 2};
struct Point p2 = {10, 20};
struct Point p3 = add_point(p1, p2);
add_assign_point(&p1, p2);

/* p1.x and p3.x are now both 11. */
/* p1.y and p3.y are now both 22. */
```

(d) (1 Punkt) Definieren Sie eine Funktion `char* find_char(char c, char* s)` die den String `s` nach dem Zeichen `c` durchsucht. Ist `c` in `s` enthalten, so soll ein Zeiger auf das erste Vorkommen von `c` in `s` zurückgegeben werden, ansonsten der `NULL`-Zeiger.[1]

```
char* s = "foo";

char* c1 = find_char('o', s);
/* c1 == s+1 */

char* c2 = find_char('x', s);
/* c2 == NULL */
```

Aufgabe 4.2 (Fingerübungen: `malloc` und `free`; 11 Punkte)

Diese Aufgabe soll in der Datei `warmup_malloc.c` bearbeitet werden. Schreiben Sie sinnvolle Unittests für jede Funktion die Sie definieren.

Jeder Speicherbereich der mit `malloc` alloziert wurde, muss bis spätestens zum Programmende wieder mit `free` freigegeben werden, sodass der Speicherbereich von weiteren `malloc`-Aufrufen wiederverwendet werden kann. Die Aufrufe von `malloc` und `free` müssen dabei nicht immer in der selben Funktion stattfinden. Wenn man es vergisst einen Speicherbereich nach Benutzung wieder freizugeben, dann nennt man das ein *Memory Leak*. Für Memory Leaks gibt es Punktabzug. Dies gilt auch für Unittests und auch für alle weitere Aufgaben und Übungsblätter.

(a) (1 Punkt) Schreiben Sie die Funktion `fib_fast` aus Übungsblatt 2 so um, dass das `cache`-Array nicht mehr fest die Länge 5000 hat und für `n > 4999` die Funktion `fib_slow` aufruft, sondern mit `malloc` ein Array von geeigneter Größe anlegt.

Sie können die Implementierung von `fib_fast` und die zugehörigen Unittests aus der Musterlösung von Übungsblatt 2 verwenden, die auf der Vorlesungsseite zu finden ist.

(b) (2 Punkte) Definieren Sie eine Funktion `char* repeat(size_t n, char* s)`, die einen neuen String zurückgibt in dem `n`-mal hintereinander `s` steht. Für `repeat(3, "foo")` soll also der String `"foofoofoo"` zurückgegeben werden.

[1] Der `NULL`-Zeiger zeigt auf die Speicheradresse 0 und wird per Konvention verwendet um Fehler zu signalisieren. Die Speicheradresse 0 ist aus diesem Grund reserviert und ein Zugriff führt direkt zu einem Segmentation Fault. `NULL` ist in der `stdlib.h` als Macro `#define NULL 0` definiert.

(c) (4 Punkte) Schreiben Sie eine Funktion

```
char* join(char** strings, size_t num_strings, char* separator)
```

die ein Array von Strings, dessen Länge und einen weiteren String `separator` als Argumente nimmt und einen *neuen* String zurückgibt in dem die im Array enthaltenen Strings nebeneinander stehen und jeweils zwei Strings durch den `separator` voneinander getrennt werden.

Zum Beispiel soll `join(strings, 4, " * ")` für

```
char* strings[] = {
  "Help!",
  "I'm split into",
  "multiple",
  "strings!"
};
```

den folgenden String zurückgeben:

```
"Help! * I'm split into * multiple * strings!"
```

(d) (4 Punkte) Schreiben Sie eine Funktion `char* find_quoted(char* s)`, die den String `s` nach Text durchsucht, der zwischen zwei Anführungszeichen enthalten ist. Bei Erfolg soll der Text der zwischen den ersten beiden Anführungszeichen gefunden wurde in einem *neuen* String zurückgegeben werden. Gibt es weniger wie zwei Anführungszeichen so soll der `NULL`-Zeiger zurückgegeben werden.

```
char* s1 = find_quoted("foo \"bar\" baz \"boo");
char* s2 = find_quoted("foo \"bar");
char* s3 = find_quoted("foo");
/* s1 is "bar". */
/* s2 and s3 are NULL. */
```

Aufgabe 4.3 (Erfahrungen; 2 Punkte)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt in der Datei `erfahrungen.txt` (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Der Zeitaufwand *muss* dabei in der ersten Zeile und in exakt dem folgenden Format notiert werden, da wir sonst nicht automatisiert eine Statistik erheben können:

```
Zeitaufwand: 3:30

<...Andere Erfahrungen...>
```

Die Angabe `3:30` steht hier für 3 Stunden und 30 Minuten.