

Hardwarenahe Programmierung

Musterlösung zu den Übungsaufgaben – 7. Dezember 2023

Aufgabe 1: Länge von Strings

Strings werden in der Programmiersprache C durch Zeiger auf **char**-Variable realisiert.

Beispiel: **char** *hello_world = "Hello,_world!\n"

Die Systembibliothek stellt eine Funktion **strlen()** zur Ermittlung der Länge von Strings zur Verfügung (**#include <string.h>**).

- (a) Auf welche Weise ist die Länge eines Strings gekennzeichnet? (1 Punkt)
- (b) Wie lang ist die Beispiel-String-Konstante "Hello,_world!\n", und wieviel Speicherplatz belegt sie? (2 Punkte)
- (c) Schreiben Sie eine eigene Funktion **int strlen (char *s)**, die die Länge eines Strings zurückgibt. (3 Punkte)

Wir betrachten nun die folgenden Funktionen (Datei: **aufgabe-1.c**):

```
int fun_1 (char *s)
{
    int x = 0;
    for (int i = 0; i < strlen (s); i++)
        x += s[i];
    return x;
}
```

```
int fun_2 (char *s)
{
    int i = 0, x = 0;
    int len = strlen (s);
    while (i < len)
        x += s[i++];
    return x;
}
```

- (d) Was bewirken die beiden Funktionen? (2 Punkte)
- (e) Schreiben Sie eine eigene Funktion, die dieselbe Aufgabe erledigt wie **fun_2()**, nur effizienter. (4 Punkte)
- (f) Von welcher Ordnung (Landau-Symbol) sind die beiden Funktionen hinsichtlich der Anzahl ihrer Zugriffe auf die Zeichen im String? Begründen Sie Ihre Antwort. Sie dürfen für **strlen()** Ihre eigene Version der Funktion voraussetzen. (3 Punkte)
- (g) Von welcher Ordnung (Landau-Symbol) ist Ihre effizientere Funktion? Begründen Sie Ihre Antwort. (1 Punkt)

Lösung

- (a) **Auf welche Weise ist die Länge eines Strings gekennzeichnet?**

Ein String ist ein Array von **chars**. Nach den eigentlichen Zeichen des Strings enthält das Array ein **Null-Symbol** (Zeichen mit Zahlenwert 0, nicht zu verwechseln mit der Ziffer '0') als Ende-Markierung. Die Länge eines Strings ist die Anzahl der Zeichen *vor* diesem Symbol.

- (b) **Wie lang ist die Beispiel-String-Konstante "Hello,_world!\n", und wieviel Speicherplatz belegt sie?**

Sie ist 14 Zeichen lang ('\n' ist nur 1 Zeichen; das Null-Symbol, das das Ende markiert, zählt hier nicht mit) und belegt Speicherplatz für 15 Zeichen (15 Bytes – einschließlich Null-Symbol / Ende-Markierung).

- (c) **Schreiben Sie eine eigene Funktion **int strlen (char *s)**, die die Länge eines Strings zurückgibt.**

Siehe die Dateien **loesung-1c-1.c** (mit Array-Index) und **loesung-1c-2.c** (mit Zeiger-Arithmetik). Beide Lösungen sind korrekt und arbeiten gleich schnell.

Die Warnung **conflicting types for built-in function "strlen"** kann normalerweise ignoriert werden; auf manchen Systemen (z. B. MinGW) hat jedoch die eingebaute Funktion **strlen()** beim Linken Vorrang vor der selbstgeschriebenen, so daß die selbstgeschriebene Funktion nie aufgerufen wird. In solchen Fällen ist es zulässig, die selbstgeschriebene Funktion anders zu nennen (z. B. **my_strlen()**).

(d) **Was bewirken die beiden Funktionen?**

Beide addieren die Zahlenwerte der im String enthaltenen Zeichen und geben die Summe als Funktionsergebnis zurück.

Im Falle des Test-Strings "Hello,_world!\n" lautet der Rückgabewert 1171 (siehe [loesung-1d-1.c](#) und [loesung-1d-2.c](#)).

(e) **Schreiben Sie eine eigene Funktion, die dieselbe Aufgabe erledigt wie `fun_2()`, nur effizienter.**

Die Funktion wird effizienter, wenn man auf den Aufruf von `strlen()` verzichtet und stattdessen die Ende-Prüfung in derselben Schleife vornimmt, in der man auch die Zahlenwerte der Zeichen des Strings aufsummiert.

Die Funktion `fun_3()` in der Datei [loesung-1e-1.c](#) realisiert dies mit einem Array-Index, Die Funktion `fun_4()` in der Datei [loesung-1e-2.c](#) mit Zeiger-Arithmetik. Beide Lösungen sind korrekt und arbeiten gleich schnell.

Bemerkung: Die effizientere Version der Funktion arbeitet doppelt so schnell wie die ursprüngliche, hat aber ebenfalls die Ordnung $\mathcal{O}(n)$ – siehe unten.

(f) **Von welcher Ordnung (Landau-Symbol) sind die beiden Funktionen hinsichtlich der Anzahl ihrer Zugriffe auf die Zeichen im String? Begründen Sie Ihre Antwort. Sie dürfen für `strlen()` Ihre eigene Version der Funktion voraussetzen.**

Vorüberlegung: `strlen()` greift in einer Schleife auf alle Zeichen des Strings der Länge n zu, hat also $\mathcal{O}(n)$.

`fun_1()` ruft in jedem Schleifendurchlauf (zum Prüfen der `while`-Bedingung) einmal `strlen()` auf und greift anschließend auf ein Zeichen des Strings zu, hat also $\mathcal{O}(n \cdot (n + 1)) = \mathcal{O}(n^2)$.

`fun_2()` ruft einmalig `strlen()` auf und greift anschließend in einer Schleife auf alle Zeichen des Strings zu, hat also $\mathcal{O}(n + n) = \mathcal{O}(n)$.

(g) **Von welcher Ordnung (Landau-Symbol) ist Ihre effizientere Funktion?**

Begründen Sie Ihre Antwort.

In beiden o. a. Lösungsvarianten – [loesung-1e-1.c](#) und [loesung-1e-2.c](#) – arbeitet die Funktion mit einer einzigen Schleife, die gleichzeitig die Zahlenwerte addiert und das Ende des Strings sucht.

Mit jeweils einer einzigen Schleife haben beide Funktionen die Ordnung $\mathcal{O}(n)$.

Aufgabe 2: Einfügen in Strings (Ergänzung)

Diese Aufgabe ist eine Ergänzung von Aufgabe 3 der Übung vom 31. Oktober 2022 um die Teilaufgaben (e), (f) und (g). Für den „Klausur-Modus“ können Sie die Teilaufgaben (a) bis (d) als „bereits gelöst“ voraussetzen.

Wir betrachten das folgende Programm ([aufgabe-2.c](#)):

```
#include <stdio.h>
#include <string.h>

void insert_into_string (char src, char *target, int pos)
{
    int len = strlen (target);
    for (int i = pos; i < len; i++)
        target[i+1] = target[i];
    target[pos] = src;
}

int main (void)
{
    char test[100] = "Hochshule_Bochum";
    insert_into_string ('c', test, 5);
    printf ("%s\n", test);
    return 0;
}
```

Die Ausgabe des Programms lautet: Hochschhhhhhhhhhh

- (a) Erklären Sie, wie die Ausgabe zustandekommt.
- (b) Schreiben Sie die Funktion `insert_into_string()` so um, daß sie den Buchstaben `src` an der Stelle `pos` in den String `target` einfügt.
Die Ausgabe des Programms müßte dann `Hochschule Bochum` lauten.
- (c) Was kann passieren, wenn Sie die Zeile `char test[100] = "Hochshule_Bochum";` durch `char test[] = "Hochshule_Bochum";` ersetzen? Begründen Sie Ihre Antwort.
- (d) Was kann passieren, wenn Sie die Zeile `char test[100] = "Hochshule_Bochum";` durch `char *test = "Hochshule_Bochum";` ersetzen? Begründen Sie Ihre Antwort.
- (e) Schreiben Sie eine Funktion `void insert_into_string_sorted (char src, char *target)`, die voraussetzt, daß der String `target` alphabetisch sortiert ist und den Buchstaben `src` an der alphabetisch richtigen Stelle einfügt. Diese Funktion darf die bereits vorhandene Funktion `insert_into_string()` aufrufen.
(4 Punkte)
Zum Testen eignen sich die folgenden Zeilen im Hauptprogramm:
- ```
char test[100] = "";
insert_into_string_sorted ('c', test);
insert_into_string_sorted ('a', test);
insert_into_string_sorted ('d', test);
insert_into_string_sorted ('b', test);
```
- Danach sollte `test[]` die Zeichenfolge `"abcd"` enthalten.
- (f) Wie schnell (Landau-Symbol in Abhängigkeit von der Länge  $n$  des Strings) arbeitet Ihre Funktion `void insert_into_string_sorted (char src, char *target)`? Begründen Sie Ihre Antwort. (1 Punkt)
- (g) Beschreiben Sie – in Worten oder als C-Quelltext –, wie man die Funktion `void insert_into_string_sorted (char src, char *target)` so gestalten kann, daß sie in  $\mathcal{O}(\log n)$  arbeitet. (3 Punkte)

## Lösung

**Bemerkung:** Die in dieser Aufgabe und ihrer Musterlösung vorkommenden Funktionen prüfen nicht, ob durch das Einfügen eines Zeichens der für den String reservierte Speicherplatz überläuft. Ein derartiges Verhalten wäre in einem „echten“ Programm ein **Fehler**, der katastrophale Folgen haben kann. Wenn dergleichen hier nicht berücksichtigt wird, dann nur, um in einer Klausur nicht den zeitlichen Rahmen zu sprengen.

- (e) **Schreiben Sie eine Funktion `void insert_into_string_sorted (char src, char *target)`, die voraussetzt, daß der String `target` alphabetisch sortiert ist und den Buchstaben `src` an der alphabetisch richtigen Stelle einfügt. Diese Funktion darf die bereits vorhandene Funktion `insert_into_string()` aufrufen.**

```
void insert_into_string_sorted (char src, char *target)
{
 int i = 0;
 while (target[i] && target[i] < src)
 i++;
 insert_into_string (src, target, i);
}
```

Die Datei `loesung-2e.c` enthält die o. a. Funktion sowie zusätzliche Tests.

- (f) **Wie schnell (Landau-Symbol in Abhängigkeit von der Länge  $n$  des Strings) arbeitet Ihre Funktion `void insert_into_string_sorted (char src, char *target)`? Begründen Sie Ihre Antwort.**

Die Funktion sucht im Array **mittels einer Schleife** nach der korrekten Position zum Einfügen des Zeichens und hat daher von sich aus  $\mathcal{O}(n)$ .

Anschließend ruft sie die Funktion `insert_into_string()` auf, die ebenfalls eine Schleife verwendet, um im Array Platz zu Einfügen zu schaffen, und daher ebenfalls  $\mathcal{O}(n)$  hat.

Es bleibt daher bei  $\mathcal{O}(n)$ .

- (g) **Beschreiben Sie – in Worten oder als C-Quelltext –, wie man die Funktion `void insert_into_string_sorted(char src, char *target)` so gestalten kann, daß sie in  $\mathcal{O}(\log n)$  arbeitet.**

In einem alphabetisch sortierten Array kann man die Suche in der Mitte beginnen und sich durch Halbieren der Intervalle an die gesuchte Position herantasten. Wegen des fortwährenden Halbierens geschieht dies in  $\mathcal{O}(\log n)$ . (Für eine derartige Antwort gäbe es in der Klausur die volle Punktzahl.)

Wenn wir allerdings anschließend für das eigentliche Einfügen die Funktion `insert_into_string()` verwenden, die dafür  $\mathcal{O}(n)$  benötigt, kommen wir insgesamt auf  $\mathcal{O}(n)$ . Ein sortiertes Einfügen in ein Array ist daher in  $\mathcal{O}(\log n)$  nicht möglich. (Wer dies bemerkt, kann zum einen während der Klausur nachfragen, wie denn die Aufgabenstellung genau gemeint ist, und sich zum anderen für die besondere Sorgfalt Zusatzpunkte verdienen.)

Die Datei `loesung-2g.c` enthält einen C-Quelltext, die den o. a. Algorithmus als Funktion implementiert. Man beachte die Behandlung des Spezialfalls, daß das einzufügende Zeichen am Ende angehängt werden muß.