

# Hardwarenahe Programmierung

## Musterlösung zu den Übungsaufgaben 6 – 28. November 2024

### 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)

### 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.

## Aufgabe 2: Iterationsfunktionen

Wir betrachten das folgende Programm (`aufgabe-2.c`):

```
#include <stdio.h>

void foreach (int *a, void (*fun) (int x))
{
    for (int *p = a; *p >= 0; p++)
        fun (*p);
}

void even_or_odd (int x)
{
    if (x % 2)
        printf ("%d_ist_ungerade.\n", x);
    else
        printf ("%d_ist_gerade.\n", x);
}
```

```
int main (void)
{
    int numbers[] = { 12, 17, 32, 1, 3, 16, 19, 18, -1 };
    foreach (numbers, even_or_odd);
    return 0;
}
```

- (a) Was bedeutet `void (*fun) (int x)`, und welchen Sinn hat seine Verwendung in der Funktion `foreach()`? (2 Punkte)
- (b) Schreiben Sie das Hauptprogramm `main()` so um, daß es unter Verwendung der Funktion `foreach()` die Summe aller positiven Zahlen in dem Array berechnet. Sie dürfen dabei weitere Funktionen sowie globale Variable einführen. (4 Punkte)

## Lösung

- (a) **Was bedeutet `void (*fun) (int x)`, und welchen Sinn hat seine Verwendung in der Funktion `foreach()`?**

`void (*fun) (int x)` deklariert einen Zeiger `fun`, der auf Funktionen zeigen kann, die einen Parameter `x` vom Typ `int` erwarten und keinen Wert zurückgeben (`void`).

Durch die Übergabe eines derartigen Parameters an die Funktion `foreach()` lassen wir dem Aufrufer die Wahl, welche Aktion für alle Elemente des Arrays aufgerufen werden soll.

- (b) **Schreiben Sie das Hauptprogramm `main()` so um, daß es unter Verwendung der Funktion `foreach()` die Summe aller positiven Zahlen in dem Array berechnet. Sie dürfen dabei weitere Funktionen sowie globale Variable einführen.**

Siehe: `loesung-2.c`

Damit die Funktion `add_up()` Zugriff auf die Variable `sum` hat, muß diese global sein und vor der Funktion `add_up()` deklariert werden.

Die Bedingung, daß nur positive Zahlen summiert werden sollen, ist durch die Arbeitsweise der Funktion `foreach()` bereits gewährleistet, da negative Zahlen als Ende-Markierungen dienen.

Wichtig ist, daß die Variable `sum` vor dem Aufruf der Funktion `foreach()` auf den Wert 0 gesetzt wird. In `loesung-2.c` geschieht dies durch die Initialisierung von `sum`. Wenn mehrere Summen berechnet werden sollen, muß dies durch explizite Zuweisungen `sum = 0` vor den Aufrufen von `foreach()` erfolgen.