

Hardwarenahe Programmierung

Musterlösung zu den Übungsaufgaben – 12. Dezember 2022

Aufgabe 1: Länge von Strings

Diese Aufgabe ist eine Neuauflage von Aufgabe 2 der Übung vom 7. November 2022, ergänzt um die Teilaufgaben (f) und (g).

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-2.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-2c-1.c](#) (mit Array-Index) und [loesung-2c-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-2d-1.c](#) und [loesung-2d-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-2e-1.c](#) realisiert dies mit einem Array-Index, Die Funktion `fun_4()` in der Datei [loesung-2e-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-2e-1.c](#) und [loesung-2e-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: Text-Grafik-Bibliothek

Schreiben Sie eine Bibliothek für „Text-Grafik“ mit folgenden Funktionen:

- **`void clear (char c)`**
 Bildschirm auf Zeichen `c` löschen,
 also komplett mit diesem Zeichen (z. B.: Leerzeichen) füllen
- **`void put_point (int x, int y, char c)`**
 Punkt setzen (z. B. einen Stern `*`) an die Stelle (x, y) „malen“)
- **`char get_point (int x, int y)`**
 Punkt lesen
- **`void display (void)`**
 das Gezeichnete auf dem Bildschirm ausgeben

Hinweise:

- Eine C-Bibliothek besteht aus (mindestens) einer `.h`-Datei und einer `.c`-Datei.
- Verwenden Sie ein Array als „Bildschirm“.
Vor dem Aufruf der Funktion `display()` ist nichts zu sehen;
alle Grafikoperationen erfolgen auf dem Array.

- Verwenden Sie Präprozessor-Konstante, z. B. `WIDTH` und `HEIGHT`,
um Höhe und Breite des „Bildschirms“ festzulegen:

```
#define WIDTH 72  
#define HEIGHT 24
```

- Schreiben Sie zusätzlich ein Test-Programm, das alle Funktionen der Bibliothek benutzt,
um ein hübsches Bild (z. B. ein stilisiertes Gesicht – „Smiley“) auszugeben.

(8 Punkte)

Lösung

Siehe die Dateien `textgraph.c` und `textgraph.h` (Bibliothek) sowie `test-textgraph.c` (Test-Programm).

Diese Lösung erfüllt zusätzlich die Aufgabe, bei fehlerhafter Benutzung (Koordinaten außerhalb des Zeichenbereichs) eine sinnvolle Fehlermeldung auszugeben, anstatt unkontrolliert Speicher zu überschreiben und abzustürzen.

Das Schlüsselwort `static` bei der Deklaration der Funktion `check_coordinates()` bedeutet, daß diese Funktion nur lokal (d. h. innerhalb der Bibliothek) verwendet und insbesondere nicht nach außen (d. h. für die Benutzung durch das Hauptprogramm) exportiert wird. Dies dient dazu, nicht unnötig Bezeichner zu reservieren (Vermeidung von „Namensraumverschmutzung“).

Man beachte die Verwendung einfacher Anführungszeichen (Apostrophe) bei der Angabe von `char`-Konstanten (`'*'`) im Gegensatz zur Verwendung doppelter Anführungszeichen bei der Angabe von String-Konstanten (String = Array von `chars`, abgeschlossen mit Null-Symbol). Um das einfache Anführungszeichen selbst als `char`-Konstante anzugeben, ist ein vorangestellter Backslash erforderlich: `'\"'` („Escape-Sequenz“). Entsprechendes gilt für die Verwendung doppelter Anführungszeichen innerhalb von String-Konstanten:
`printf("Your_name_is:_%s\\", name);`

Aufgabe 3: PBM-Grafik

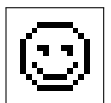
Bei einer PBM-Grafikdatei handelt es sich um ein abgespeichertes C-Array von Bytes (`uint8_t`), das die Bildinformationen enthält:

- Die Datei beginnt mit der Kennung `P4`, danach folgen Breite und Höhe in Pixel als ASCII-Zahlen, danach ein Trennzeichen und die eigentlichen Bilddaten.
- Jedes Bit entspricht einem Pixel.
- Nullen stehen für Weiß, Einsen für Schwarz.
- MSB first.
- Jede Zeile des Bildes wird auf ganze Bytes aufgefüllt.

Viele Grafikprogramme können PBM-Dateien öffnen und bearbeiten. Der Anfang der Datei (Kennung, Breite und Höhe) ist auch in einem Texteditor lesbar.

Beispiel (`aufgabe-3.pbm`):

```
P4  
14 14  
<Bilddaten>
```



In dem untenstehenden Programmfragment (`aufgabe-3.c`) wird eine Grafik aus Textzeilen zusammengesetzt, so daß man mit einem Texteditor „malen“ kann:

```

#include <stdio.h>

/* ... */

int main (void)
{
    pbm_open (14, 14, "test.pbm");
    pbm_line ("");
    pbm_line ("XXXXXX");
    pbm_line (" X X");
    pbm_line (" X X");
    pbm_line (" X X");
    pbm_line (" X XX XX X");
    pbm_line (" X X X X");
    pbm_line (" X X");
    pbm_line (" X X X X");
    pbm_line (" X X X X");
    pbm_line (" X XXXX X");
    pbm_line (" X X");
    pbm_line (" XXXXX");
    pbm_line ("");
    pbm_close ();
    return 0;
}

```

Ergänzen Sie das Programmfragment so, daß es eine Datei `test.pbm` erzeugt, die die Grafik enthält.

Das Programm soll typische Benutzerfehler abfangen (z. B. weniger Zeilen als in `pbm_open` angegeben), keine fehlerhaften Ausgaben produzieren oder abstürzen, sondern stattdessen sinnvolle Fehlermeldungen ausgeben.

Zum Vergleich liegt eine Datei `aufgabe-3.pbm` mit dem gewünschten Ergebnis bei, und die Datei `aufgabe-3.png` enthält dasselbe Bild.

(10 Punkte)

Lösung

Die Datei `loesung-3.c` enthält eine richtige Lösung. Man beachte die Aufrufe der Funktion `error()` im Falle von falscher Benutzung der Bibliotheksfunktionen. Weitere Erklärungen finden Sie als Kommentare im Quelltext.

Die Datei `loesung-3f.c` enthält eine falsche Lösung. (Beide Dateien unterscheiden sich nur in Zeile 46.) Dieses Programm speichert die Bits in den Bytes von rechts nach links (LSB first). Richtig wäre von links nach rechts (MSB first). Das erzeugte Bild ist dementsprechend fehlerhaft.