

Hardwarenahe Programmierung / Angewandte Informatik

Musterlösung zu den Übungsaufgaben – 7. November 2016

Aufgabe 1: Strings

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

Wir betrachten die folgende Funktion (Datei: [aufgabe-1.c](#)):

```
int fun_1 (char *s1, char *s2)
{
    int result = 1;
    for (int i = 0; s1[i] && s2[i]; i++)
        if (s1[i] != s2[i])
            result = 0;
    return result;
}
```

- (a) Was bewirkt die Funktion?
- (b) Welchen Sinn hat die Bedingung „s1[i] && s2[i]“ in der **for**-Schleife?
- (c) Was würde sich ändern, wenn die Bedingung „s1[i] && s2[i]“ in der **for**-Schleife zu „s1[i]“ verkürzt würde?
- (d) Schreiben Sie eine eigene Funktion, die dieselbe Aufgabe erledigt wie `fun_1()`, nur effizienter.

Lösung

- (a) **Was bewirkt die Funktion?**

Sie vergleicht zwei Strings miteinander bis zur Länge des kürzeren Strings und gibt bei Gleichheit 1 zurück, ansonsten 0.

Mit anderen Worten: Die Funktion prüft, ob zwei Strings bis zur Länge des kürzeren übereinstimmen, und gibt bei Gleichheit 1 zurück, ansonsten 0.

Die Funktion prüft insbesondere **nicht** zwei Strings auf Gleichheit, und sie ist **nicht** funktionsgleich zur Standard-Bibliotheksfunktion `strcmp()`.

- (b) **Welchen Sinn hat die Bedingung „s1[i] && s2[i]“ in der for-Schleife?**

Die Bedingung prüft, ob *bei einem der beiden Strings* die Ende-Markierung (Null-Symbol) erreicht ist. Falls ja, wird die Schleife beendet.

- (c) **Was würde sich ändern, wenn die Bedingung „s1[i] && s2[i]“ in der for-Schleife zu „s1[i]“ verkürzt würde?**

In diesem Fall würde nur für `s1` geprüft, ob das Ende erreicht ist. Wenn `s1` länger ist als `s2`, würde `s2` über sein Ende hinaus ausgelesen. Dies kann zu Lesezugriffen auf Speicher außerhalb des Programms und damit zu einem Absturz führen („Speicherzugriffsfehler“, „Schutzverletzung“).

- (d) **Schreiben Sie eine eigene Funktion, die dieselbe Aufgabe erledigt wie `fun_1()`, nur effizienter.**

Die Effizienz lässt sich steigern, indem man die Schleife abbricht, sobald das Ergebnis feststeht. Es folgen drei Möglichkeiten, dies zu realisieren.

Erweiterung der Schleifenbedingung:

```
int fun_2 (char *s1, char *s2)
{
    int result = 1;
    for (int i = 0; s1[i] && s2[i] && result; i++)
        if (s1[i] != s2[i])
            result = 0;
    return result;
}
```

Verwendung von **return**:

```
int fun_3 (char *s1, char *s2)
{
    for (int i = 0; s1[i] && s2[i]; i++)
        if (s1[i] != s2[i])
            return 0;
    return 1;
}
```

Die nebenstehende Lösung unter Verwendung von **break** ist zwar ebenfalls richtig, aber länger und weniger übersichtlich als die beiden anderen Lösungen.

Die Datei `loesung-1.c` enthält ein Testprogramm für alle o. a. Lösungen. Das Programm testet nur die offensichtlichsten Fälle; für den Einsatz der Funktionen in einer Produktivumgebung wären weitaus umfassendere Tests erforderlich.

Das Testprogramm enthält String-Zuweisungen wie z. B. `s2 = "Apfel"`. Dies funktioniert, weil wir damit einen Zeiger (`char *s2`) auf einen neuen Speicherbereich ("Apfel") zeigen lassen. Eine entsprechende Zuweisung zwischen Arrays (`char s3[] = "Birne"; s3 = "Pfirsich";`) funktioniert *nicht*.

```
int fun_4 (char *s1, char *s2)
{
    int result = 1;
    for (int i = 0; s1[i] && s2[i]; i++)
        if (s1[i] != s2[i])
        {
            result = 0;
            break;
        }
    return result;
}
```

Aufgabe 2: Text-Grafik-Bibliothek

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

- **void clear (char c)**
Bildschirm auf Zeichen `c` löschen
- **void put_point (int x, int y, char c)**
Punkt setzen
- **char get_point (int x, int y)**
Punkt lesen
- **void display (void)**
Inhalt des Arrays auf dem Bildschirm ausgeben

Hinweise:

- Eine C-Bibliothek besteht aus (mindestens) einer `.h`-Datei und einer `.c`-Datei.
- Verwenden Sie ein Array als „Bildschirm“.
- Verwenden Sie Präprozessor-Konstante, z. B. `WIDTH` und `HEIGHT`, um Höhe und Breite des „Bildschirms“ festzulegen.
- Schreiben Sie zusätzlich ein Test-Programm, das alle Funktionen der Bibliothek benutzt, um ein hübsches Bild (z. B. ein stilisiertes Gesicht – „Smiley“) auszugeben.

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);`