Programmieren 1 Übung:
Funktionen mit String-Parameter, vordef. String-Funktionen

Klaus Kusche

1.) Umrechnen von Elektronik-Farbcodes

Gesucht ist ein Programm, das mit den *3 Farben (als Text)* der Markierungsringe
auf der Befehlszeile aufgerufen wird und den Wert des Widerstandes ausgibt.
Den Toleranz-Farbring lassen wir unberücksichtigt
(d.h. die 3 Farben entsprechen *2 Ziffern und der Zehnerpotenz*),
ebenso Gold und Silber für negative Zehnerpotenzen.

Verwende dazu eine *Funktion*, die mit *einer Farbe* (als Text, d.h. **`const char *`**)
aufgerufen wird und die entsprechende *Ziffer* (als **`int`**) zurückgibt.

In dieser Funktion wirst du ein *Array von 10 Stringkonstanten*
(initialisiert mit den *Farben in der richtigen Reihenfolge*) brauchen.
Vergleiche die übergebene Farbe der Reihe nach mit den Farben im Array
(mit welcher vordefinierten Funktion vergleicht man Strings?), bis du sie findest:
Der *Index* ist die Ziffer, die dieser Farbe entspricht.

Findest du die Farbe nicht, soll das Programm *mit einer Fehlermeldung enden*
(keinen Fehler-Returnwert an das Hauptprogramm zurückgeben, sondern
gleich in der Funktion die Fehlermeldung ausgeben und das Programm abbrechen).

Zum Ausrechnen der Zehnerpotenz im Hauptprogramm darfst du
die *vordefinierte Potenzfunktion* (**`pow`** aus **`math.h`**) verwenden.

2.) Text ersetzen

Versuche, folgende **String-Funktion** zu schreiben:

```
char *strrepl(char dest[], const char src[],
              const char oldStr[], const char newStr[])
```

Die Funktion soll **`src`** nach **`dest`** *kopieren* und dabei alle Vorkommen von **`oldStr`**
durch **`newStr`** *ersetzen* (**`src`** soll dabei *unverändert* bleiben!).
Der *Returnwert* soll **`dest`** sein. Da es keinen Parameter für die Größe von **`dest`** gibt,
nehmen wir ohne Prüfung an, dass **`dest`** groß genug für das Ergebnis ist (Pfui!).

Die Groß- und Kleinschreibung wird beim Vergleich beachtet.
Ob **`oldStr`** in **`src`** als alleinstehendes Wort oder innerhalb eines Wortes vorkommt,
ist egal: In beiden Fällen wird er durch **`newStr`** ersetzt.

Wenn **`oldStr`** leer ist, wird *nichts ersetzt*: **`src`** wird *unverändert* nach **`dest`** kopiert.
Wenn **`newStr`** leer ist, werden alle Vorkommen von **`oldStr`** im Ergebnis *gelöscht*
(durch nichts ersetzt).

Vorgeschlagene Vorgehensweise:

- Du brauchst *2 Pointer*, die in den String **`src`** zeigen:
  Einen auf die *"aktuelle Position"*, bis zu der **`src`** schon verarbeitet ist,
  und einen auf die nächste *Fundstelle* von **`oldStr`**.

- Fange zuerst den Sonderfall ab, dass **oldStr** leer ist und **src** nur nach **dest** kopiert wird, und merk dir dabei auch gleich die Länge von **oldStr**.
- Sonst initialisiere **dest** auf einen *leeren* String (wie?)
  und setze deine aktuelle Position in **src** auf den *Anfang* von **src**.
- Mach eine *Schleife*, die *pro Vorkommen* von **oldStr** in **src** *einen Umlauf* macht:
  - *Suche* das erste Vorkommen von **oldStr** *ab der aktuellen Position* in **src** (welche Stringfunktion kannst du dafür verwenden, was liefert sie?).
  - Wenn **oldStr** im Rest von **src** nicht mehr vorkommt: *Beende* die Schleife.
  - Hänge den *Ausschnitt* von **src** zwischen der aktuellen Position und der Fundstelle von **oldStr** an **dest** an.
    Tipp: Verwende dazu **strncat**; wie berechnest du die *Anzahl der Zeichen* zwischen aktueller Position und Fundstelle?
  - Hänge **newStr** an **dest** an (wieder mit einer Stringfunktion).
  - Setze die *aktuelle Position* in **src** unmittelbar *hinter* das gefundene Vorkommen von **oldStr** (wie viele Zeichen hinter der Fundstelle ist das?).
- Nach der Schleife musst du noch den gesamten *Rest* von **src** ab der aktuellen Position an **dest** *anhängen*.

Verwende nach Möglichkeit die vordefinierten Stringfunktionen!

Schreib dazu ein **Hauptprogramm**:

oldStr und **newStr** werden beim Programmstart auf der *Befehlszeile* angegeben.
Die Texte, in denen gesucht und ersetzt wird, werden vom *DOS-Fenster* gelesen, und zwar *zeilenweise*, bis zum Eingabeende oder Programmabbruch.
Für jede gelesene Zeile wird **strrepl** aufgerufen und das Ergebnis ausgegeben.

Hinweise:

- Du darfst im **main** *zwei Strings fixer Länge* (Konstante definieren, nimm 4096) verwenden (für die Eingabezeile und für das Ergebnis von **strrepl**) und brauchst *nicht auf Überlauf zu prüfen*.

  Würde man die Funktion "ordentlich" programmieren, bräuchte man einen zusätzlichen Parameter für die maximale Länge des Ergebnisses.
- Das *Einlesen* einer ganzen Zeile vom DOS-Fenster in ein **char**-Array **zeile** funktioniert mit **fgets(zeile, sizeof(zeile), stdin)** .

  Das liest genau eine Zeile vom DOS-Fenster nach **zeile**, einschließlich dem **'\n'** am Ende und einem **'\0'**.

  Ctrl/Z (Windows) oder Ctrl/D (Linux) beendet die Eingabe (zeigt "Dateiende" an).

  fgets schreibt man normalerweise direkt in die Bedingung einer **while**-Schleife: Es liefert **zeile**, also einen von **NULL** verschiedenen Wert, als *Returnwert*, wenn es erfolgreich war, und **NULL**, wenn das Lesen schiefgegangen ist (z.B. am Dateiende).
- Wenn du einen Text mit Zwischenräumen oder Tabulatoren als ein einziges Wort von der Befehlszeile einlesen willst, musst du ihn in **"** … **"** einschließen.
  Ein leeres Wort kannst du auf der Befehlszeile mit **""** eingeben.