Programmieren C: Strings, vordef. Stringfunktionen: Textersetzungsfunktion

Klaus Kusche

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 die Vorkommen von **oldStr** im Ergebnis *gelöscht* (durch nichts ersetzt).

Das Beispiel kann entweder unter *Verwendung von vordefinierten Stringfunktionen* oder *händisch zeichenweise ohne Stringfunktionen* gelöst werden.

Vorgeschlagene Vorgehensweise bei Verwendung von **Stringfunktionen:**

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

Vorgeschlagene Vorgehensweise bei **direkter Lösung:**

- Du brauchst einen Zeiger auf das *nächste zu prüfende Zeichen* in **`src`** und einen Zeiger auf das *nächste zu schreibende Zeichen* in **`dest`**.
- Mach eine *Schleife*, die **`src`** bis zur *Ende-Markierung* durchläuft:
    - Wenn das *aktuelle Zeichen* in **`src`** *nicht gleich* dem *ersten Zeichen* von **`oldStr`** ist, dann wird das Zeichen einfach nach **`dest`** *kopiert*, und beide Pointer rücken *eins weiter*.
    - Sonst musst du **`src`** ab der aktuellen Position mit einer weiteren Schleife *zeichenweise* mit **`oldStr`** *vergleichen* (achte darauf, dass dein Vergleich auch dann sauber endet, wenn beide Strings *gleichzeitig enden*, d.h. wenn **`oldStr`** am Ende von **`src`** vorkommt!):
        - Kommt **`oldStr`** an dieser Stelle in **`src`** *komplett* vor, dann setze den **`src`**-Pointer auf das Zeichen *unmittelbar nach dem Vorkommen* und *kopiere* **`newStr`** mit einer Schleife *zeichenweise* nach **`dest`**.
        - Sonst *kopiere* das aktuelle **`src`**-Zeichen nach **`dest`** und rücke mit beiden Pointern *eins weiter*.
- Vergiss nicht, nach der Schleife auch **`dest`** ordnungsgemäß zu *beenden*.

Schreib dazu ein **Hauptprogramm**:

`oldStr` und **`newStr`** werden beim Programmstart auf der *Befehlszeile* angegeben.
Die Texte, in denen gesucht und ersetzt wird, werden vom *Terminal* gelesen, und zwar *zeilenweise*, bis zum Eingabeende (End of File) 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 Ersetzen-Funktion "ordentlich" programmieren, bräuchte man einen *zusätzlichen Parameter* für die maximale Länge des Ergebnisses und Prüfungen, damit auch nicht mehr Zeichen im Ergebnis gespeichert werden.
- Wenn du einen *Text mit Zwischenräumen oder Tabulatoren* als ein *einziges* Wort auf der *Befehlszeile* angeben willst, musst du ihn in **`"`** … **`"`** einschließen.
  Ein *leeres Wort* (*"ersetzte durch nichts"*) gibt man auf der Befehlszeile mit **`""`** ein.
- Das Einlesen einer ganzen Zeile vom Terminal in das **`char`**-Array **`zeile`** funktioniert mit **`fgets(zeile, sizeof(zeile), stdin)`** .

  Das liest genau eine Zeile vom Terminal 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).