Programmieren C: Zeilenweises I/O, Strings: Wortersetzung

Klaus Kusche

Dein Programm wird mit *zwei Worten* auf der Befehlszeile aufgerufen,
gefolgt von *keinem, einem oder zwei Filenamen*.

Bei *keinem* Filenamen soll es von **`stdin`** lesen und auf **`stdout`** schreiben,
bei *einem* Filenamen von diesem File lesen und auf **`stdout`** schreiben,
und bei *zwei* Filenamen ist der erste der Input-File und der zweite der Output-File.

Es soll den Input auf den Output kopieren und dabei
alle Vorkommen des ersten Wortes durch das zweite Wort ersetzen,
egal, ob das Wort in einer Zeile gar nicht, einmal oder mehrmals vorkommt.

Die *Groß- und Kleinschreibung* wird beim Vergleich *beachtet*.
Ob das erste Wort als *alleinstehendes Wort* oder *innerhalb eines längeren Wortes* vorkommt,
ist egal: *In beiden Fällen* wird es durch das zweite Wort *ersetzt*.

Lösungsidee:

- Der Input wird *zeilenweise* verarbeitet.
- In jeder Zeile suchen wir das *erste Vorkommen* des gesuchten Wortes
  (dafür gibt es eine String-Funktion, verwenden!).
- An der Stelle, an der das gesuchte Wort anfängt,
  beenden wir den bisherigen Text der Zeile (siehe unten!)
  und *geben den Teil bis dorthin aus* (ohne Zeilenvorschub!).
- Gleich anschließend *geben wir das Ersatzwort aus*.
- Dann *suchen wir wieder* nach einem Vorkommen des gesuchten Wortes,
  und zwar *ab dem Zeichen* der Zeile, das auf das gerade bearbeitete Vorkommen
  des Wortes *folgt* (d.h. wir überspringen das Wort).
- Das wiederholen wir, bis das Wort im Rest der Zeile *nicht mehr vorkommt*.
  Dann wird der *gesamte Rest der Zeile ausgegeben*
  (ab der Stelle, wo die letzte Suche begonnen hat).

Tipps und Hinweise:

- Nimm das in der Stunde besprochene File-I/O-Beispiel als Ausgangsbasis!
- *Prüfe alle deine I/O-Operationen* auf Fehler und gib *ordentliche Fehlermeldungen* aus!
- Du darfst fix eine *maximale Input-Zeilenlänge* (groß, z.B. 4096) vorgeben
  und sollst mit einer *Fehlermeldung* abbrechen, wenn sie überschritten wird.
- Du wirst mit **`char`**-Pointern arbeiten müssen.
- **Überlege:**
  Wenn du aus *dem vorderen Teil* eines Strings *einen eigenen String machen* möchtest
  und das Zeichen, vor dem der neue String enden soll, *nicht mehr brauchst*:
  Wie kannst du den String an dieser Stelle am einfachsten "teilen"?
  (du darfst die gelesene Zeile *modifizieren*, du solltest sie *nicht umkopieren*)
- Überlege, ob irgendein *Sonderfall im Programm-Aufruf* abgefangen werden muss.

Zusatzaufgabe: Stringfunktion zum Ersetzen

Das Ergebnis stückchenweise auszugeben, mag zwar in diesem Fall
vielleicht die effizientere Lösung sein, aber resultiert in einem Stück Code,
das nicht für andere Zwecke wiederverwendbar ist.

Wir wollen daher eine ***universelle "String-Ersetzen-Funktion"*** schreiben:

```
char *strrepl(char dest[], const char src[], int destLen,
              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.

destLen ist die *Größe* von **dest**. Passt das Ergebnis nicht in **dest** hinein,
soll die Funktion einen *leeren String* in **dest** speichern und **NULL** als *Returnwert* liefern.

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).

Unser *Hauptprogramm* ruft dann nur diese Funktion auf
und gibt *je nach Returnwert* entweder deren *Ergebnis* oder eine *Fehlermeldung* aus.

Die erste Idee ist, die *Logik von oben* nachzubauen, nur dass die einzelnen Teilstücke
nicht ausgegeben, sondern mit **strcat** / **strncat** in **dest** *aneinandergehängt* werden:

- Fange zuerst den *Sonderfall* ab, dass **oldStr** *leer* ist und **src** nur nach **dest**
  kopiert wird, und merke dir dabei auch gleich die *Länge* von **oldStr**.
- Du brauchst wie oben *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**.
- 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*.

Vergiss nicht, vor jedem **strcpy**, **strcat** und **strncat** eine *Längenprüfung* zu machen!

Diese erste Variante ist *noch nicht sonderlich effizient*,
u.a. weil **strcat** bzw. **strncat** jedesmal wieder das Ende des Strings suchen muss.

Als Alternative käme eine Lösung in Frage, die **src** *zeichenweise* durchläuft und *völlig ohne Stringfunktionen* auskommt:

- 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 **dest** ordnungsgemäß zu *beenden* und vor jedem Kopieren zu prüfen, ob *noch Platz* in **dest** ist.