Programmieren 1 Übung: Strings

Klaus Kusche

Einfache String-Funktionen

Versuche, **folgende String-Funktionen** zu schreiben
(es müssen nicht alle sein, suche dir je nach Schwierigkeit ein paar aus):

- **`char *strrepeat(char *dest, char c, int cnt);`**

  Ganz leicht: Speichere **`cnt`** Mal den Buchstaben **`c`** in **`dest`**.

  Ergebnis: **`dest`**

- **`char *strtextcpy(char *dest, const char *src);`**

  Leicht: Kopiere **`src`** nach **`dest`**, aber *ohne* Zwischenräume,
  d.h. nur die Zeichen, für die **`isspace(c)`** als Ergebnis **`false`** liefert.

  Ergebnis: **`dest`**

- **`char *strappend(char *dest, const char *src);`**

  Mittelleicht: Hänge **`src`** hinten an **`dest`** an.
  Tipp: Zuerst das *Ende* von **`dest`** suchen, dann normal kopieren.

  Ergebnis: **`dest`**

- **`char *strrev(char *dest, const char *src);`**

  Mittelleicht: Kopiere **`src`** umgedreht (von hinten nach vorne) nach **`dest`**.
  Tipp: Suche zuerst das Ende von **`src`**, und kopiere dann rückwärts,
  bis du wieder am Anfang bist.

  Ergebnis: **`dest`**

- **`char *strmaxcpy(char *dest, int size, const char *src);`**

  Mittelleicht: Kopiere **`src`** nach **`dest`**, aber *maximal* **`size-1`** Zeichen.
  Tipp: Deine Schleife wird *zwei* Bedingungen prüfen müssen.

  Ergebnis: **`dest`**

- **`char *strrevchr(const char *str, char c);`**

  Mittelleicht: Returniere einen Pointer auf das *letzte* Vorkommen von **`c`** in **`str`**
  (oder **`NULL`**).
  Tipp: *Nicht* zuerst das Ende suchen und dann von dort rückwärts das Zeichen **`c`**,
  sondern gleich beim Vorwärts-Durchlauf bis zum Ende
  das jeweils letzte Vorkommen von **`c`** merken!

- **`char *stranychr(const char *str, const char *charlist);`**

  Mittelleicht: Returniere einen Pointer auf das erste Vorkommen
  irgendeines Zeichens aus **`charlist`** in **`str`** (oder **`NULL`**).
  Tipp: **`str`** zeichenweise durchlaufen und das aktuelle Zeichen
  nacheinander mit *allen* Zeichen von **`charlist`** vergleichen.

- **`char *strsearch(const char *haystack, const char *needle);`**

Mittelschwer: Returniere einen Pointer auf das erste Vorkommen von **`needle`** in **`haystack`** (oder **`NULL`**).
Tipp: Vergleiche der Reihe nach für jede Anfangsposition in **`haystack`** die Zeichen von **`needle`** mit den entsprechenden aufeinanderfolgenden Zeichen in **`haystack`**.

- **`char *strtrim(char *dest, const char *src);`**

  Sehr schwer: Kopiere **`src`** nach **`dest`** (Ergebnis: **`dest`**), aber

  - *streiche* alle Zwischenräume, Tabs, Zeilenenden, … (alle Zeichen, für die **`isspace(c)`** das Ergebnis **`true`** liefert) *am Anfang und am Ende* weg …
  - und ersetze alle aufeinanderfolgenden **`isspace(c)`**-Zeichen mittendrin durch einen *einzigen* Zwischenraum.

Hinweise:

- Verwende nach Möglichkeit *Pointer-Arithmetik* und **`*`** anstelle von Index und **`[ ]`**.
- Du sollst außer **`isspace`** keine **vordefinierten String-Funktionen** verwenden!
- *Vergiss nicht, dein Ergebnis mit* **`'\0'`** *zu beenden, wenn es ein neu kopierter String ist!*

Schreib dazu je Funktion ein Hauptprogramm, das die jeweilige Funktion zum Test mit einem von der Befehlszeile eingelesenen String aufruft und das Ergebnis ausgibt.

Hinweise:

- Du darfst für das Ergebnis einen String fixer Länge verwenden und brauchst *nicht auf Überlauf zu prüfen*.

  Würde man die Funktionen "ordentlich" programmieren, bräuchten einige einen zusätzlichen Parameter für die Länge des Ergebnisses.

- 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.

Zusatzaufgabe zum Üben von **`malloc`**:

Lege bei den ersten fünf Funktionen den *Ergebnis-String dynamisch* mit **`malloc`** in genau der benötigten Größe an. Bei **`strappend`** wird **`dest`** ein reiner Eingabe-Parameter (der erste der beiden Strings, die zusammengehängt werden sollen), bei den anderen Funktionen fällt der **`dest`**-Parameter ersatzlos weg.

Schlägt das **`malloc`** fehl, sollen die Funktionen keine Fehlermeldung ausgeben, sondern einfach **`NULL`** als Ergebnis liefern.

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.
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).
Ob **oldStr** in **src** als alleinstehendes Wort oder innerhalb eines Wortes vorkommt, ist egal: In beiden Fällen wird er durch **newStr** ersetzt.

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

Vorgeschlagene Vorgehensweise bei Verwendung von **Stringfunktionen:**

- 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**.
- 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**.
- 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**.
  - Wenn **oldStr** nicht mehr vorkommt: *Beende* die Schleife.
  - Hänge den *Ausschnitt* von **src** zwischen der aktuellen Position und der Fundstelle an **dest** an. (*Tipp:* Verwende dazu **strncat**; wie berechnest du die *Anzahl der Zeichen* zwischen aktueller Position und Fundstelle?).
  - Hänge **newStr** an **dest** an.
  - 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 *zeichenweise* mit **`oldStr`** *vergleichen* (achte darauf, dass dein Vergleich auch dann sauber endet, wenn beide Strings *gleichzeitig enden*, d.h. **`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`** *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*.

Hinweise:

- **Verwende nach Möglichkeit die vordefinierten Stringfunktionen!**

Schreib dazu ein **Hauptprogramm**: Die Strings für **`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 oder Programmabbruch. Für jede Zeile wird **`strrepl`** aufgerufen und das Ergebnis angezeigt.

Hinweise:

- Du darfst im **`main`** zwei Strings fixer Länge verwenden (für die Eingabezeile und für das Ergebnis) und brauchst *nicht auf Überlauf zu prüfen*.

  Würde man die Funktion "ordentlich" programmieren, bräuchte man einen zusätzlichen Parameter für die maximale Länge des Ergebnisses.
- 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.
- 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`**, incl. dem **`'\n'`** am Ende und einem **`'\0'`**. Ctrl/Z (Windows) oder Ctrl/D (Linux) beendet die Eingabe (Dateiende).

  `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).