Programmieren C: Strings: Einfache String-Funktionen

Klaus Kusche

Hinweise für die folgenden Funktionen:

- Verwende nach Möglichkeit *Pointer-Arithmetik* und ***** anstelle von Index und **[ ]**, auch deine *Schleifenvariablen* sollten *Pointer* sein.

  In meiner Musterlösung findest Du für jede Funktion
  die Standard-Lösung mit Pointer und eine einfachere Lösung mit Index.

- Du sollst außer **`isspace`** ***keine*** **vordefinierten String-Funktionen** verwenden!

- *Vergiss nicht, deinen Ergebnis-String zu beenden, wenn es ein neu erstellter String ist!*

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 1: Suche *zuerst das Ende* von **`src`**, und *kopiere dann rückwärts*,
  bis du wieder am Anfang bist.
  Tipp 2: Eine *Schleife mit 2 Schleifenvariablen* hilft!

  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.

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

Bei der Funktion **`strrev`** könntest Du auch ein Hauptprogramm schreiben, das die *komplette Befehlszeile* (außer dem Programmnamen) *umgedreht* ausgibt. Dazu werden *alle Worte der Befehlszeile von hinten nach vorne* durchlaufen, und jedes einzelne Wort wird umgedreht ausgegeben.

Hinweis zum Testen:

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.

Hinweis / Zusatzaufgabe: Längenprüfung

Du darfst im ersten Anlauf im **`main`** für das Ergebnis einen *String fixer Länge* verwenden und brauchst in den Funktionen *nicht auf Überlauf zu prüfen*.

Würde man die Funktionen "ordentlich" programmieren, bräuchten einige einen *zusätzlichen Parameter* für die Länge des Ergebnisses und eine *Überprüfung*, ob das Ergebnis in den Zielstring *passt*.

Ich habe das für eine Funktion beispielhaft gemacht: Siehe **`strnrev`** statt **`strrev`**.

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.