AIK Programmieren 1 Übung: Strings, Funktionen, Pointer, …

Klaus Kusche

Das erste Programm ist einfach (keine Funktionen, keine Pointer, nur Strings),
das zweite deutlich schwerer (lange Funktion, Rechnen mit Pointern, …)!

1.) Wort erraten

Wir schreiben ein Programm, bei dem der Benutzer buchstabenweise ein Wort erraten muss (so wie früher im Fernsehen: "Ich möchte das '**e**' !").

Das Programm soll sich wie folgt verhalten:

- Als Erstes sucht sich das Programm *zufällig* aus einer *fixen Liste von Wörtern* das zu erratende Wort aus. Es ermittelt und speichert auch gleich dessen *Länge*.

  Erinnere dich: Was musst du tun, damit dein Programm bei jedem Aufruf ein *anderes* zufälliges Wort aus der Liste wählt?

  Verwende für die Wortliste ein *Array von Strings* (egal ob ein zweidimensionales **char**-Array oder ein Array von Zeigern auf **char** – beides funktioniert gleich, das zweite ist üblicher), das du gleich beim Anlegen *mit fixen Worten initialisierst*!

  Die *Anzahl* der Strings im Array, d.h. die Größe des Arrays (und damit den Bereich, in dem die Zufallszahl für die Auswahl des zu erratenen Wortes liegen muss), sollst du *berechnen* und *nicht* durch händisches Abzählen der Strings in der Initialisierungsliste ermitteln (welches Konstrukt verwendet man dafür?).

- Außer der Wortliste wirst du eine *String-Variable* für die *Eingabe* und eine zweite für das *angezeigte Wort* mit den '*' brauchen.

  Du darfst für beide eine fixe *Maximallänge* (Konstante definieren!) vorgeben.

  In dem String, den der Benutzer vom zu erratenden Wort angezeigt bekommt, wird am Anfang einmal ein String (was braucht der?) mit *lauter* '*' *gespeichert* (genau so viele, wie das zufällig ausgewählte, zu erratende Wort Buchstaben hat!).

- Dann passiert in einer *Schleife* immer wieder Folgendes:
  - Der Benutzer bekommt das angezeigt, was vom Wort schon bekannt ist (beim ersten Mal eben nur die '*').
  - Dann gibt er einen Text ein (nimm **scanf**, denn **fgets** hängt ein **'\n'** an). Das kann entweder ein *einzelner Buchstabe oder ein ganzes Wort* sein.
  - Hat er ein ganzes Wort (*mehr als 1 Zeichen* lang) eingegeben, so wird das eingegebene Wort mit dem zu erratenden Wort *verglichen*.
    - Ist es gleich, hat der Benutzer gewonnen.
    - Sonst wird "Falsch!" angezeigt, und der Benutzer muss den nächsten Versuch machen.
  - Hat er nur ein *einzelnes Zeichen* eingegeben, passiert Folgendes:
    - Das eingegebene Zeichen wird *mit jedem einzelnen Buchstaben* des zu erratenden Wortes verglichen (*Schleife*!).

An genau den Stellen, wo der eingegebene Buchstabe im gesuchten Wort *vorkommt*, wird im angezeigten Wort das '*' durch den richtigen Buchstaben *ersetzt*.

- Dann wird das angezeigte Wort mit dem gesuchten Wort *verglichen*.

  Falls es jetzt gleich ist (d.h. falls der Benutzer den letzten noch fehlenden Buchstaben erraten hat), hat der Benutzer auch gewonnen, sonst geht es normal mit dem nächsten Versuch weiter.

- Wenn der Benutzer das Wort erraten hat, soll angezeigt werden, *wie viele Versuche* der Benutzer gebraucht hat (jede Eingabe, egal ob ein Buchstabe oder ein ganzes Wort, egal ob richtig oder falsch, zählt als ein Versuch, also einfach in der Schleife mitzählen).

  Dann endet das Programm.

Hinweise:

- Für die Länge eines Strings und den Vergleich zweier Strings sollst du die entsprechenden *String-Funktionen* verwenden.

  Es ist sinnvoll, die Länge des zu erratenden Wortes nur einmal am Anfang auszurechnen und in einer Variable zu speichern.

- Zur Vereinfachung kümmern wir uns nicht darum, einen Großbuchstaben auch für den entsprechenden Kleinbuchstaben zu erkennen oder umgekehrt bzw. großschreibungsunabhängig zu vergleichen: Die Worte werden intern komplett klein geschrieben gespeichert, und es werden daher auch nur eingegebene Kleinbuchstaben als passend erkannt.
- Weiters verzichten wir zur Vereinfachung auf die Längenprüfung: Du darfst eine fixe Maximalgröße für das Eingabewort, die Worte in der Wortliste und das Ausgabewort vorgeben. Gibt der Benutzer mehr ein, darf das Programm abstürzen.
- *Achtung:* Auch wenn eine String-Variable nur ein einziges Zeichen enthält, ist sie trotzdem ein String und kein Einzelzeichen (**char**): Wenn man nur das eine Zeichen allein will, muss man eben das erste Zeichen des Strings auswählen!

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

Vorgeschlagene Vorgehensweise:

- 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** (welche Stringfunktion kannst du dafür verwenden, was liefert sie?).
  - 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*.

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* (Konstante definieren, nimm 4096) verwenden (für die Eingabezeile und 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).