AIK Programmieren 1 Übung: Einfache String-Funktionen

Klaus Kusche

1.) Palindrome erkennen

Palindrome sind Worte oder Sätze, die *von vorne nach hinten und von hinten nach vorne gelesen gleich* sind, z.B. “Rentner”. “Lagerregal” oder “Reliefpfeiler”. Wir wollen ein Programm schreiben, das mit *einem Wort auf der Befehlszeile* aufgerufen wird und ausgibt, ob dieses Wort ein Palindrom ist oder nicht.

Dazu schreiben wir eine *Funktion*, die den *zu prüfenden String* übergeben bekommt und ein *ja/nein-Ergebnis* zurückliefert (ja … Palindrom, nein … kein Palindrom).
Dazu vergleichen wir mit einer *Schleife* den *ersten* mit dem *letzten* Buchstaben (zum Feststellen der *Wortlänge* darfst du eine vordefinierte *Stringfunktion* verwenden), den *zweiten* mit dem *vorletzten* usw..
Finden wir dabei verschiedene Buchstaben, ist das Wort kein Palindrom.
Haben wir bis zur Wortmitte nur gleiche Buchstabenpaare, ist das Wort ein Palindrom.

Schreib dazu ein *Hauptprogramm*, das die Funktion mit dem ersten Wort der Befehlszeile aufruft und je nach Ergebnis ausgibt, ob dieses Wort ein Palindrom ist oder nicht.

Zusatzaufgabe 1: Referenz-Parameter

Erweitere die Funktion um zwei **`int`**-Referenzparameter.
Ist das Wort kein Palindrom, soll die Funktion in diesen beiden Parametern die *Position* (den Index) der beiden Zeichen speichern, die nicht zusammenpassen.
Ist das Wort ein Palindrom, lässt die Funktion die beiden Parameter unverändert.

Das Hauptprogramm soll so erweitert werden, dass es im nein-Fall auch die beiden Positionen und die beiden Zeichen an diesen Positionen *ausgibt*.

Zusatzaufgabe 2: Groß- und Kleinschreibung sowie Satzzeichen entfernen

Erstens sollten wir beim Vergleichen *Groß- und Kleinschreibung ignorieren*, und zweitens müssen wir alle *Zwischenräume, Satzzeichen usw. überspringen*, wenn wir auch Satz-Palindrome wie “Ein Neger mit Gazelle zagt im Regen nie” erkennen wollen.

Dazu schreiben wir eine *Funktion*, die mit *zwei Strings* (Quelle und Ziel) aufgerufen wird, den Quellstring händisch *Zeichen für Zeichen* in den Zielstring kopiert, und dabei erstens alle Buchstaben *in Kleinbuchstaben verwandelt* und zweitens alle Zeichen, die kein Buchstabe sind, *weglässt*. Als Returnwert soll die Funktion den Zielstring liefern.

Die Funktion soll auch prüfen, ob der Zielstring *zu kurz* für das Ergebnis ist.
Dazu bekommt sie als *dritten Parameter* übergeben, wie viel Platz im Zielstring ist.
Passt das Ergebnis nicht in den Zielstring, soll eine Fehlermeldung ausgegeben werden.
Die Funktion soll trotzdem den Zielstring returnieren, mit so viel Text wie Platz hat.

Im Hauptprogramm definieren wir uns eine *String-Variable* (definiere eine Konstante für die maximale Länge) und *kopieren* das Wort von der Befehlszeile mit unserer Funktion in diesen String. Dann machen wir unseren Palindrom-Test so wie in der vorigen Version des Programmes auf diesem String.

Hinweise:

- Um auf der Befehlszeile einen Text mit Zwischenräumen als ein *einziges* Wort einzugeben, musst du ihn in **" "** einschließen!
- Verwende für die Prüfung auf Buchstaben und die Verwandlung in Kleinbuchstaben die *vordefinierte Funktionen*.
- Was darfst du beim händischen Kopieren in deinen String nicht vergessen?

2.) Commandline umdrehen

Wir wollen ein Programm schreiben, das den auf der Befehlszeile angegebenen Text *buchstabenweise umgekehrt* (von hinten nach vorne) ausgibt.

Dazu müssen wir die Worte auf der Befehlszeile *von hinten nach vorne* durchgehen und *jedes einzelne Wort umgedreht ausgeben*.

Das *Umdrehen eines einzelnen Wortes* soll dabei in einer *eigenen Funktion* geschehen:

- Die Funktion bekommt *zwei Strings übergeben:* Das Array, in dem die Funktion den *Ergebnis-String* speichern soll, und den *umzudrehenden String*.

  Die Funktion soll also die Buchstaben des zweiten Strings in umgekehrter Reihenfolge (von hinten nach vorne) in den ersten String hineinkopieren.
- Da wir nicht wissen, wie lange der umzudrehende String ist und ob er überhaupt in den Ergebnis-String hineinpasst, muss der Funktion beim Aufruf auch die *Größe des Ergebnis-Arrays* übergeben werden (ist es zu klein, soll einfach nur der Teil des Quelltextes hineinkopiert werden, der hineinpasst!).
- Damit wir unsere Umdreh-Funktion auch direkt im **`printf`** aufrufen können, soll das Ergebnis-Array nicht nur als Parameter übergeben werden, sondern auch als *Returnwert* zurückkommen.

Hinweise:

- Verwende zum buchstabenweisen Kopieren eine Schleife, die mit *zwei Variablen gleichzeitig* zählt: Eine beginnt beim *letzten Buchstaben des Quelltextes* und läuft Richtung Anfang des Quelltextes, die andere gibt die *aktuelle Position im Ergebnis* an und zählt von 0 aufwärts.

  Die Schleife läuft, bis entweder das vorderste Zeichen des Quelltextes kopiert *oder* das *vorletzte* (wieso nicht das letzte?) Zeichen des Ergebnisses gefüllt wurde.
- Du darfst in der ersten Version des Programmes normale **`int`**'s als Index im Quell- und Ergebnis-String verwenden. Den Umbau auf Pointer machen wir erst später.
- Zur Ermittlung der Länge des Quelltextes darfst du die vordefinierte Funktion **`strlen`** aus **`string.h`** verwenden.
- Was musst du mit dem Ergebnis-String noch tun, *nachdem* du alle Zeichen des Quell-Strings hineinkopiert hast (d.h. nach der Schleife)?

Schreib dazu ein *Hauptprogramm*, das einen String fixer Länge für das Ergebnis anlegt, diese Funktion von hinten nach vorne *für jedes einzelne Wort* der Befehlszeile aufruft, und ihr Ergebnis ausgibt. Definiere für die maximale Stringlänge eine Konstante!

Zusatzaufgabe: Umbau auf Pointer

Wenn die Funktion funktioniert, kannst du sie *auf Pointer umbauen*.
Dabei ändert sich nur das Innenleben der Funktion,
Hauptprogramm und Aufruf bleiben gleich:

Die Schleife arbeitet mit zwei Pointern, einer zeigt auf das aktuelle Zeichen
im Quelltext und einer auf das entsprechende Zeichen im Ergebnis.

Überlege: Wie prüfst du, ob du schon am Ende des Ergebnis-Strings bist?

3.) Umwandlung Text ==> Dezimalzahl

Gesucht ist eine Funktion, die das macht, was **`atoi`** bisher für uns gemacht hat:
Sie bekommt einen *Text*, der eine ganze Dezimalzahl enthält, als *Parameter*,
verwandelt ihn in einen **`int`**, und liefert diesen als *Returnwert*.

- Wir gehen dazu den String von vorne nach hinten *Zeichen für Zeichen* durch
  (woran erkennst du, dass du alle Zeichen des Strings verwandelt hast?).
- Bei jedem Zeichen prüfen wir, ob es eine *Ziffer* ist, verwandeln es in den
  entsprechenden *Ziffernwert* (das haben wir am Anfang des Jahres einmal
  besprochen: Wie berechnet man aus dem ASCII-Code die Zahl zu einer Ziffer?),
  und rechnen diesen *zum bisherigen Ergebnis dazu* (wie?).

Das *Hauptprogramm* soll diese Funktion der Reihe nach für alle Worte der Befehlszeile
aufrufen und ihr Ergebnis ausgeben.

Hinweise:

- Als Zusatzaufgabe könntest du noch ein **`'-'`** an der *ersten* Stelle eines Wortes
  (und nur dort!) richtig behandeln. Am einfachsten ist es, wenn man sich
  das *Vorzeichen* merkt und *nach* der Umwandlung zum Ergebnis dazurechnet,
  und zwar am einfachsten als *Multiplikationsfaktor* (**`1`** oder **`-1`**).
- Wenn ein Eingabewort ein Zeichen enthält, das *keine Ziffer* ist,
  sollte eine *Fehlermeldung* (mit dem Zeichen!) ausgegeben werden.
  Die Umwandlung wird beendet und die bisher umgewandelte Zahl zurückgegeben.
- Um festzustellen, ob ein Zeichen eine *Ziffer* ist,
  solltest du die entsprechende *vordefinierte Funktion* verwenden
  (oder eine **`if`**-Bedingung mit zwei Teilen).