Programmieren 1 Übung: Strings

Klaus Kusche

1.) Umwandlung Text ==> Dezimalzahl und zurück

Als ersten Schritt wollen wir das machen, was **`atoi`** bisher für uns gemacht hat:
Einen Text, der eine ganze Dezimalzahl enthält, in einen **`int`** verwandeln.

- Unser Programm soll alle auf der Eingabezeile angegebenen Worte der Reihe nach in einen **`int`** verwandeln und diesen ausgeben.
- Wir gehen dazu jedes einzelne Wort von vorne nach hinten *Zeichen für Zeichen* durch (woran erkennst du, dass du alle Zeichen eines Wortes 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?).

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* ausgegeben werden.
- Um festzustellen, ob ein Zeichen eine Ziffer ist, solltest du die entsprechende *vordefinierte Funktion* verwenden (oder eine **`if`**-Bedingung mit zwei Teilen).

Der zweite Schritt ist die *umgekehrte Umwandlung*, von einem **`int`** in einen *Text*, so wie es **`printf`** macht. Dieser Schritt ist etwas schwieriger, wenn der Ergebnis-String *nur so lang sein soll, wie es zur Darstellung der jeweiligen Zahl nötig* ist.

Es gibt mehrere Varianten:

- Die Ziffern von vorne nach hinten berechnen. Das ist kompliziert und langsam.
- Zuerst die Länge feststellen, dann die Ziffern von hinten nach vorne gleich an die richtige Stelle in das Ergebnis speichern. Das ist auch kompliziert und langsam.
- Die Ziffern mit einer rekursiven Funktion von vorne nach hinten berechnen. Das ist die einfachste und eleganteste Variante, aber das können wir noch nicht.
- Die Ziffern *von hinten nach vorne* berechnen, *zwischenspeichern*, und wenn man fertig ist und weiß, wie viele es sind, *an die richtige Stelle des Ergebnisses kopieren*, oder die Ziffern *von hinten nach vorne* berechnen, in *verkehrter* Reihenfolge von links nach rechts speichern, und nachher *umdrehen*. So wollen wir es machen.

Wir erweitern das Programm aus dem ersten Schritt:
Bei jedem einzelnen Wort verwandeln wir den umgewandelten **`int`** gleich wieder zurück in einen String und geben den String (mit **`printf`** und **`%s`**) aus.

- Überlege dir, *wie groß* du die String-Variable für das Ergebnis anlegen musst: Die größte in einem **`int`** darstellbare Zahl ist in etwa 2 Milliarden (10 Ziffern), das Vorzeichen muss auch Platz haben, und einen gültigen String müssen wir daraus ebenfalls noch machen.
- *Berechne* in einer Schleife immer wieder *die hinterste Ziffer* aus der umzuwandelnden Zahl, streiche sie aus der Zahl weg, verwandle sie in das entsprechende *ASCII-Zeichen* (wie?), und speichere dieses *im Ergebnis-String* (wahlweise von ganz hinten nach vorne oder von vorne nach hinten).

  Wiederhole das, bis von der ursprünglichen Zahl nichts mehr übrig ist (0).
- Korrigiere dann den String:

  Wenn du *von hinten nach vorne* gespeichert hast, musst du die Zeichen nach links *schieben*, damit die erste Ziffer am Anfang des Strings steht.

  Wenn du die Ziffern *von vorne nach hinten* gespeichert hast, musst du sie *umdrehen* (eine Schleife zum Umdrehen eines Strings stand einmal auf einer Folie).

  Beides solltest du mit einer Schleife selbst programmieren, ohne Verwendung vordefinierter Funktionen.
- *Achtung:* Das Ergebnis muss *immer mindestens eine Stelle* haben, auch wenn die ursprüngliche Zahl schon 0 ist!

Hinweise:

- Es ist wieder klug, das *Vorzeichen* ganz am Anfang zu prüfen und zu speichern und dann die Zahl *ohne Vorzeichen zu verwandeln*.
- Achte darauf, dass dein Ergebnis ein *gültiger String* ist (was muss man dazu am Ende tun?)!

2.) 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.

Im einfachsten Fall 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 verschiedene Buchstaben, ist das Wort kein Palindrom (gib aus, welche Zeichen an welchen Stellen nicht zusammenpassen!). Haben wir bis zur Wortmitte nur gleiche Buchstabenpaare, ist das Wort ein Palindrom.

Zusatzaufgabe:

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.

Wir definieren uns dazu eine *String-Variable* und *kopieren* das Wort von der Befehlszeile händisch *Zeichen für Zeichen* in diesen String. Dabei kopieren wir *nur die Buchstaben* und *verwandeln* sie auch gleich in *Kleinbuchstaben*. Dann machen wir unseren Palindrom-Test so wie in der ersten 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*.
- Prüfe, ob das Wort auf der Befehlszeile *zu lang* für deinen String ist, und brich in diesem Fall mit einer Fehlermeldung ab.
- Was darfst du beim händischen Kopieren in deinen String nicht vergessen?