Institut für Informatik
Daniel Neugebauer

Wintersemester 2019/20
Tag 5

Hardwarenahe Programmierung

Gruppe 6 (Stefan)

Ab der heutigen Übung programmieren Sie wieder ausschließlich in C. In dieser Übung liegt Ihr Fokus auf dem Umgang mit Strings in C-Programmen.

Wichtig: *Zu einigen der Aufgaben heute stellen wir Unittests. Achten Sie darauf, dass Ihre Abgaben diese erfüllen!*

Aufgabe 1 *String Konkatenierung*

(a) Schreiben Sie eine Funktion `my_strcat`, welche zwei Strings entgegen nimmt, diese konkateniert und in einem Ausgabestring speichert. Wählen Sie *eine* der beiden folgenden Funktionssignaturen aus, um Ihre Funktion entsprechend zu implementieren:

entweder `void my_strcat(char *string1, char *string2, char *result)`
oder `char* my_strcat(char *string1, char *string2)`

Der Ausgabestring (`result` bzw. Rückgabewert) darf hierbei nicht länger als notwendig sein. Ob Sie den Speicher für den Ausgabestring innerhalb oder außerhalb der Funktion `my_strcat` reservieren, ist Ihnen überlassen.

Verwenden Sie keine Funktionen aus *string.h* außer `strlen()`!

(b) Schreiben Sie ein Hauptprogramm, welches die Funktion Ihrer Funktion demonstriert.

Aufgabe 2 *Großbuchstaben*

Schreiben Sie ein Programm, das folgende Funktionen enthält. Vereinfachend dürfen Sie in der gesamten Aufgabe davon ausgehen, dass Eingabestrings nur Buchstaben enthalten.

(a) eine Funktion `to_upper(char *string)`, die alle Kleinbuchstaben des gegebenen Strings in die entsprechenden Großbuchstaben ändert (aus "Kaffee" wird zum Beispiel "KAFFEE"). Überlegen Sie sich, wie der C-Datentyp `char` intern gespeichert ist, und wie Sie dies nutzen können, um diese Änderung elegant zu lösen.

Input:	K	a	f	f	e	e	\0
Output:	K	A	F	F	E	E	\0

(b) eine Funktion `int compare_ignorecase(char *string1, char *string2)`, die alle Zeichen in beiden übergebenen Strings in die entsprechenden Großbuchstaben umwandelt, und die 1 zurückgibt, falls beide umgewandelten Strings dann gleich sind, und 0 sonst.

(c) Wir haben Ihnen Unit-Tests und ein Makefile zur Verfügung gestellt. Sie können die Tests mit dem Befehl '`make run`' bauen und ausführen. Testen Sie Ihre beiden Funktionen aus (a) und (b) mit den bereitgestellten Unit-Tests und stellen Sie sicher, dass alle erfüllt sind. Sie sollten die Tests nicht verändern! Schauen Sie sich außerdem den Aufbau des Makefiles an, damit Sie in Zukunft selbst Makefiles erstellen können, um Ihren Kompilier- und Testprozess zu vereinfachen.

Aufgabe 3 *Tokenizer*

In dieser Aufgabe sollen Sie ein Programm schreiben, das Texte in Token (Wörter, die durch Leerzeichen oder Sonderzeichen getrennt sind) aufteilen kann, und das die gesammelten Token zählen kann. Wörter bestehen aus Buchstaben und/oder Ziffern. Als Sonderzeichen gelten hier "." "," "!" und "?". Sie dürfen davon ausgehen, dass keine anderen Zeichen außer Buchstaben, Ziffern und diesen Sonderzeichen in Eingaben vorkommen.

Input: "`...Kein Text? Ein Text!`"

Tokens:	"`Kein`"	"`Text`"	"`Ein`"
Häufigkeit:	1	2	1

(a) Implementieren Sie alle in der header-Datei *count_all.h* deklarierten Funktionen, und beachten Sie dabei die Kommentare in der Datei. Ihre Implementierung muss alle Unit-Tests in *count_all_tests.ts* erfüllen.

(b) Schreiben Sie ein main-Programm, das den Benutzer auffordert einen Text (max. 80 Zeichen) einzugeben, das diesen Text in Tokens aufteilt und schließlich jedes Token mit dessen Häufigkeit im Eingabetext ausgibt. Sie können die bereitgestellte Datei *main.c* als Grundlage verwenden.