Institut für Informatik
Prof. Dr. Stefan Conrad

Sommersemester 2020
Übung 2

Hardwarenahe Programmierung
Gruppe 05 (Florian)

In dieser Übung liegt der Fokus auf dem Umgang mit Strings in C-Programmen.

Wichtig: *Denken Sie wie immer daran, Ihre Lösungen im ILIAS hochzuladen und den Test im ILIAS zu absolvieren!*

Aufgabe 0 *String Konkatenierung*

(a) Schreiben Sie eine Funktion `strings_verbinden`, welche zwei Strings entgegen nimmt, diese konkateniert und in einem Ausgabestring speichert. Implementieren Sie die Funktion in einer Datei mit Namen `strings_verbinden.c`. Wählen Sie eine der beiden folgenden Funktionssignaturen:

entweder `void strings_verbinden(char zusammen[], char string1[], char string2[])`
oder `void strings_verbinden(char *zusammen, char *string1, char *string2)`

Sie dürfen hierbei annehmen, dass das `char`-Array mit Namen `zusammen`, in dem Sie den String speichern, ausreichend groß ist.

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

(b) Verwenden Sie die bereitgestellten Tests, um zu demonstrieren, dass Ihre Funktion wie gefordert funktioniert. Verwenden Sie hierzu den Kommandozeilenbefehl `make run`, der die Tests baut, kompiliert und ausführt.

Aufgabe 1 *Caesar-Verschlüsselung*

Schreiben Sie ein Programm, das folgende Funktionen enthält. Vereinfachend dürfen Sie in der gesamten Aufgabe davon ausgehen, dass Eingabestrings ausschließlich aus Großbuchstaben bestehen.

(a) eine Funktion `void caesar(char *klartext)`, die einen String mit der Caesar-Chiffre codiert: Jeder Buchstabe soll um 2 Stellen im Alphabet nach links verschoben werden (so wird z.B. “C” zu “A” und “B” zu “Z”). Überlegen Sie sich, wie der C-Datentyp `char` intern gespeichert ist, und wie Sie dies nutzen können, um diese Verschiebung elegant zu lösen.

Input:	A	S	S	E	M	B	L	Y	\0
Output:	Y	Q	Q	C	K	Z	J	W	\0

(b) eine Funktion `int encode_and_compare(char *clearstring, char *string_to_encode)`, welche die beiden eingegebenen Strings vergleicht und 1 zurück gibt, falls `clearstring` gleich der verschlüsselten Fassung von `string_to_encode` ist, 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 2 *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.