FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Algorithmen & Datenstrukturen ILV
Übungstest - Gruppe C

Hinweise zum Autograding

Die Übungsaufgaben werden in ***stack.cpp*** gelöst und in *main.cpp* getestet. Alle anderen Dateien bleiben unberührt! Zeile 2 in *main.cpp* - #define TEST 1 - bestimmt ob Tests ausgeführt werden, oder nicht. 0 bedeutet, dass nicht getestet wird. Für die Abgabe muss die Konstante immer den Wert 1 aufweisen.

K.O. Kriterien
- Programm **kompiliert** auf Github Classroom **nicht**!
- Der Quellcode wurde von anderen Studierenden **offensichtlich kopiert** (0 Punkte für alle)
- Es wurden Dateien modifiziert, die nicht modifiziert werden dürfen
- In der *main* Funktion dürfen **keine eigenen return statements** gesetzt werden. Sonst werden keine Testfälle ausgeführt.
- Es dürfen **keine Eingabeaufforderungen (scanf, gets,...)** von der Konsole erfolgen (Timeout)

Aufgabe 1

Stack erstellen und Elemente einfügen
(4 Punkte)

Ziel ist es, einen Stack mit Hilfe einer einfach verketteten Liste zu implementieren, die dem LIFO Prinzip folgt. Es gibt keine Kapazität, der Stack kann beliebig groß werden. Der Stack selbst wird durch eine Struktur repräsentiert (*struct _stack*). Elemente, die auf dem Stack eingefügt werden, greifen auf die Struktur *element* zurück. Das Funktionsprinzip eines Stack ist in Abbildung 1 dargestellt.

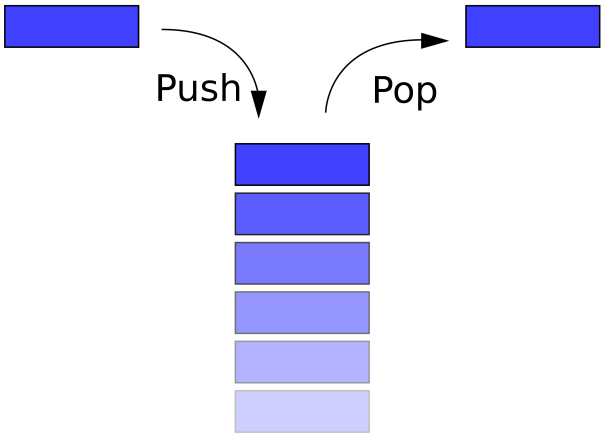

Abbildung 1: LIFO Prinzip eines Stack. Quelle: wikipedia.org

Vervollständigen Sie zunächst folgende Funktionen im Grundgerüst:

*stack *s_init();* Reserviert Speicher für einen Stack, initialisiert die Variablen und liefert eine Adresse auf den neuen Stack zurück. (2 Punkte)

*void s_push(stack *st, const char *data);* Legt ein neues Element auf dem Stack ab. (2 Punkte)

Aufgabe 2

Probleme beheben
(2 Punkte)

In den Funktionen *s_pop* und *s_top* befinden sich Fehler. Beheben Sie diese. Eine Beschreibung der Funktionen findet sich im Source Code.

Aufgabe 3

Add All
(2 Punkte)

Schreiben Sie eine Funktion *void s_addAll(stack *st, char **words, int size)*, welche alle Wörter in umgekehrter Reihenfolge aus dem Array *words* auf den Stack *st* pushed. D.h. das erste Wort aus der Liste soll oben aufliegen. Die Variable *size* gibt die Größe des Arrays an.

Aufgabe 4

Anwendung
(2 Punkte)

Schreiben Sie eine Funktion *int s_isBalanced(char *expression)*, welche einen geklammerten, mathematischen Ausdruck (*expression*) auf Ausgewogenheit hinsichtlich öffnender und schließender Klammern überprüft. Im Fehlerfall soll 0, ansonsten 1 zurückgeliefert werden. Verwenden Sie dazu einen Stack.

Beispiele:
(1 + 3) - 1 => balanciert (return 1)
(1 + 3 * 3 => unbalanciert (return 0)

Hinweis zur Vorgehensweise: Scannen Sie die Expression von Start bis Ende. Jedes Mal, wenn eine öffnende Klammer gelesen wird, "pushen" Sie diese auf den Stack. Jedes Mal, wenn Sie eine schließende Klammer lesen, "poppen" Sie ein Element des Stacks. Wenn der Stack am Ende leer ist, wissen Sie, dass der Klammerausdruck balanciert war.