Modulare Programmierung

Aufgabe 1:

Implementieren Sie eine Datenstruktur `stack_t` mittels modularer Programmierung. Die Struktur soll folgende Operationen unterstützen:

- `stack_t *createStack();` erzeugt einen leeren Stack
- `char isEmpty(stack_t *s);` prüft, ob der Stack `s` leer ist
- `void push(stack_t *s, float value);` legt den Wert `value` auf den Stack `s`
- `float top(stack_t *s);` liefert das zuletzt eingefügte Element des Stacks `s`
- `void pop(stack_t *s);` entfernt das zuletzt eingefügte Element vom Stack `s`
- `char getError(stack_t *s);` liefert den Inhalt der Fehlervariablen
- `void destroyStack(stack_t *s);` zerstört den Stack und gibt belegten Speicherplatz frei

Der Stack soll bei Bedarf automatisch vergrößert werden, so dass er beliebig viele Float-Werte speichern kann. Wenn auf einem leeren Stack ein `top` oder `pop` ausgeführt wird, soll eine Fehlervariable gesetzt werden.

Schreiben Sie ein Programm, dass einen arithmetischen Ausdruck in Umgekehrter Polnischer Notation einliest, mit Hilfe des Stacks auswertet und das Ergebnis anzeigt.

Aufgabe 2:

- `heap_t *createHeap();` erzeugt einen leeren Heap
- `char insert(heap_t *h, int val);` fügt den Wert `val` in den Heap `h` ein
- `int minimum(heap_t *h);` liefert das minimale Element des Heaps `h`
- `char extractMin(heap_t *h);` entfernt das minimale Element aus dem Heap `h`
- `char getError(heap_t *h);` liefert den Inhalt der Fehlervariablen des Heaps `h`
- `char* toString(heap_t *h);` liefert den Inhalt der Heaps `h` als Zeichenkette
- `void destroyHeap(heap_t *h);` zerstört den Heap `h` und gibt belegten Speicherplatz frei

Der Heap soll bei Bedarf automatisch vergrößert werden, so dass er beliebig viele `int`–Werte speichern kann. Wenn `minimum` auf einem leeren Heap ausgeführt wird, soll eine Fehlervariable gesetzt werden.

Erstellen Sie einen Testtreiber und Testfälle, um die Korrektheit Ihrer Implementierung nachzuweisen.