Übungen zur Vorlesung
GRUNDLAGEN DER INFORMATIK I
Studiengang Elektrotechnik
SS 03

AG Betriebssysteme FB3
Kirsten Berkenkötter

Ausgabe: 29.4.2003
Besprechung: 8.5.2003

Übungszettel 1

Implementierung eines Stacks

Wie bereits in der Vorlesung gezeigt wurde, kann man sich einen unbegrenzten Stack (Stapel) bauen, indem man ihn als Liste implementiert (ein begrenzter Stack kann als Array implementiert werden). Die Größe des Stacks ist dann nur durch den verfügbaren Speicher begrenzt, nicht aber durch feste Arraygrenzen bei der Deklaration. Wer nicht mehr weiss, wie Listen funktionieren, sollte sich nochmal die Unterlagen aus dem letzten Semester ansehen!

Die Datentypen für den Stack

Der Stack soll Datentypen folgenden Aussehens aufnehmen:

```
typedef struct
{
//Nummer und beschreibender Text werden gespeichert
int nummer;
char text[20];
} data_t;
```

Ein Element des Stacks soll folgendes Aussehen haben:

```
struct stackelement_t
{
//Daten und Zeiger auf das vorhergehende Element werden gespeichert
data_t *inhalt;
struct stackelement_t *prev;
};
```

Da man bei einem Stack alle Elemente, wie der Name schon sagt, stapelt, kann immer nur auf das oberste Element zugegriffen werden. Deshalb wird der Zeiger auf das Vorgängerelement gespeichert und nicht auf den Nachfolgenden. Der Stack-Handle hat deshalb auch als einziges Element den Zeiger auf das oberste Element des Stapels:

```
typedef struct
{
//Zeiger auf das oberste Element
struct stackelement_t *top;
} stack_handle_t;
```

Um sich lästige Tipparbeit zu sparen, wird noch ein Zeiger auf den Stack-Handle als eigener Typ definiert:

```
typedef stack_handle_t *stack_t;
```

stack_t ist also ein Zeiger auf ein Stack-Handle und braucht nicht mehr ständig mit dem Zeiger-Operator * angegeben werden.

Überprüfung von Zeigern mit *assert*

Eine weitere Vereinfachung ist die Verwendung der Funktion *assert*, der ihr als Parameter einen Zeiger übergeben könnt. Wenn dieser Zeiger *NULL* ist, wird das Programm abgebrochen und eine Fehlermeldung ausgegeben (Art des Fehlers, Name der C-Datei und Zeile in der Datei, an der der Fehler aufgetreten ist). Auf diese Art und Weise spart man sich viele *if*-Abfragen, da ja, wie aus der Listenprogrammierung bereits bekannt, immer abgefragt werden muss, ob die Parameter gültige Werte besitzen und die Zeiger nicht ins nichts führen. Für die Verwendung von *assert* muss die Bibliothek <*assert.h*> eingebunden werden.

Vorgegebene Funktionen

Ausser den Datentypen sind die Funktionen *isEmpty* und *peek* vorgegeben. *isEmpty* überprüft, ob der Stack leer ist oder nicht, *peek* gibt das oberste Datenelement zurück, ohne dieses jedoch vom Stack zu holen.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
/*——————————————————————————————————————*/
/* Funktion, die überprüft, ob der Stack leer ist
 * Parameter: stackt_s, ein Stack-Handle-Zeiger
 * Rückgabewert: 0, wenn Stack leer, 1, wenn Stack nicht leer */
int isEmpty(stack_t s)
{
   //überprüfen, ob gültiger Stack-Handle vorliegt
   assert(s);
   //Abfrage, wenn Stack leer, Rückgabe 0, sonst 1
   if(s->top == NULL)
      return 0;
   else return 1;
}
/*——————————————————————————————————————*/
/* Funktion, mit der die Daten des obersten Elements geholt werden, ohne
 * dieses zu löschen
 * Parameter: stack_t s, Stack_Handle-Zeiger
 * Rückgabewert: die Daten des obersten Elements */
data_t peek(stack_t s)
{
   //Überprüfen ob s und s->top gültig sind
   assert(s);
   assert(s->top);

   //Rückgabe der Daten, es muss nichts gelöscht werden
   return *(s->top->inhalt);
}
```

Aufgabe 1: Erzeugen eines Stacks

Schreibt eine Funktion zum Erzeugen eines Stacks. Sie soll keine Parameter erhalten und den erzeugten Stack zurückgeben. Denkt an die Reservierung des Speichers mit *malloc*!

stack_t createStack();

Aufgabe 2: Ein Datenelement auf den Stack legen

Schreibt eine Funktion, um neue Daten auf dem Stack abzulegen. Die Parameter sind ein zuvor erzeugter Stack und ein Datenelement. Es gibt keinen Rückgabewert. Denkt an die Überprüfung der Zeiger mit *assert* und an den Sonderfall, in dem der Stack noch leer ist (die Funktion *isEmpty* sollte hier verwendet werden) .

void push(stack_t s, data_t element);

Aufgabe 3: Ein Datenelement vom Stack legen

Schreibt eine Funktion, um ein Element vom Stack zu holen. Im Gegensatz zu *peek* wird das Element vom Stack gelöscht. Als Parameter wird ein zuvor erzeugter Stack übergeben, der Rückgabewert ist das oberste Datenelement. Denkt an die Überprüfung der Zeiger mit *assert* und die Freigabe des Speicherplatzes mit *free*!

data_t pop(stack_t s);