Lösungen zur Vorlesung
GRUNDLAGEN DER INFORMATIK I
Studiengang Elektrotechnik
SS 03

AG Betriebssysteme FB3
Kirsten Berkenkötter

Lösung Übungszettel 1

Die angegebene Lösung ist das komplette Stack-Programm mit einer *main*-Funktion zum Ausprobieren. Die einzelnen Aufgaben des Übungszettels können anhand der Funktionsnamen zugeordnet werden!

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

/*————————————————————————————————————*/

//ein Typ für die Daten
typedef struct
{
   //Nummer und beschreibender Text werden gespeichert
   int nummer;
   char text[20];
} data_t;

//ein Typ für ein Element des Stacks
struct stackelement_t
{
   //Daten und Zeiger auf das vorhergehende Element werden gespeichert
   data_t *inhalt;
   struct stackelement_t *prev;
};

//ein Typ für ein Stack-Handle, Verwaltung von Stacks
typedef struct
{
   //Zeiger auf das oberste Element
   struct stackelement_t *top;
} stack_handle_t ;

//ein Typ Zeiger auf Stack-Handle (damit man nicht so viel tippen muss)
typedef  stack_handle_t *stack_t;
```

```
/*————————————————————————————————————————*/

/* 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, die einen Stack erzeugt
 * Parameter: -
 * Rückgabewert: Stack-Handle-Zeiger auf neu erzeugten Stack */
stack_t createStack()
{
   //Variable für neuen Stack
   stack_t s;

   //Speicher reservieren und prüfen, ob es geklappt hat
   s = (stack_t) malloc (sizeof(stack_t));
   assert(s);

   //wenn ja, Zeiger auf oberstes Element NULL setzen, da noch kein
   //Element vorhanden ist
   s->top = NULL;

   //neuen Stack zurückgeben
   return s;
}

/*————————————————————————————————————————*/

/* Funktion, die ein Element auf den Stack legt
 * Parameter: stack_t s, der Stack, auf den das Element soll
```

```
 *          data_t element, das Element
 * Rückgabewert: - */
void push(stack_t s, data_t element)
{
   //Variable für das neue Stack-Element
   struct stackelement_t *temp;

   //überprüfen, ob der Stack gültig ist, wenn ja, Platz für neues Element
   //reservieren
   //prüfen, ob Platz reserviert werden konnte
   assert(s);
   temp = (struct stackelement_t *) malloc(sizeof(struct stackelement_t));
   assert(temp);

   //Platz für Daten reservieren und überprüfen, ob es geklappt hat
   temp->inhalt = (data_t *) malloc(sizeof(data_t));
   assert(temp->inhalt);

   //setzten und Daten kopieren
   memcpy((void *)temp->inhalt,(void *)&element, sizeof(data_t));

   //erster Fall: Stack ist bis jetzt leer
   if (isEmpty(s) == 0)
   {
      //das Vorgängerelement ist also NULL und das neue Element das neue
      //oberste Element
      temp->prev = NULL;
      s->top = temp;
   }
   //zweiter Fall: Stack ist nicht leer
   else
   {
      //bisheriges oberstes Element wird Vorgänger des neuen obersten Elements
      temp->prev = s->top;
      s->top = temp;
   }
}

/*——————————————————————————————————————————————————————*/

/* Funktion, die Daten vom Stack holt
 * Parameter: stack_t s, Stack-Handle-Zeiger
```

```
 * Rückgabwert: die Daten des obersten Elements */
data_t pop(stack_t s)
{
   //Variable für ein Stackelement und die Daten
   struct stackelement_t *temp;
   data_t datatemp;

   //Überprüfen, dass s und s->top gültig sind
   assert(s);
   assert(s->top);

   //das bisherige oberste Element in Hilfsvariable speichern (für free())
   //das bisher vorletzte Element wird neues oberstes Element
   temp = s->top;
   s->top = temp->prev;

   //die Daten in Hilfsvariable speichern
   datatemp = *(temp->inhalt);

   //den mit malloc() belegten Speicherplatz frei machen, das oberste
   //Element ist damit gelöscht
   free(temp->inhalt);
   free(temp);

   //Rückgabe der Daten des obersten Elements
   return datatemp;
}

/*——————————————————————————————————————*/

/* 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);
```

```
}

/*————————————————————————————————————————*/

/* Main-Function zum Ausprobieren des Stacks */

int main()
{
   //Variablen für den Stack und Datenelemente
   stack_t meinstack;
   data_t daten;
   data_t holen;

   //Stack erzeugen
   meinstack = createStack();

   //Daten erzeugen und auf den Stack packen
   daten.nummer=10001;
   strcpy(daten.text, "Schraube");
   push(meinstack,daten);

   //Daten erzeugen und auf den Stack packen
   daten.nummer=10022;
   strcpy(daten.text, "Nagel");
   push(meinstack,daten);

   //oberstes Element anzeigen lassen, ohne es zu löschen
   holen = peek(meinstack);
   printf("%d %s\n", holen.nummer, holen.text);

   //oberstes Element vom Stack holen und anzeigen
   holen = pop(meinstack);
   printf("%d %s\n", holen.nummer, holen.text);

   //oberstes Element vom Stack holen und anzeigen
   holen = pop(meinstack);
   printf("%d %s\n", holen.nummer, holen.text);

   //an dieser Stelle tritt ein Fehler auf, weil keine
   //Daten mehr da sind
   //assert zeigt eine Fehlermeldung an
   holen = pop(meinstack);
   printf("%d %s\n", holen.nummer, holen.text);
}
```