Benutzerdefinierte Datentypen

0) Strukturen – Einfach verkettete Listen/LIFO-Prinzip

Gegeben sei folgende Struktur

```
struct element {
     char b;
     struct element *next;
};
```

Mit Hilfe dieser Struktur soll nun eine einfach verkettet Liste realisiert werden, die nach dem Stapel-Prinzip Last-In-First-Out arbeitet. Dazu benötigen Sie drei Funktionen

- Ablegen eines Buchstabens `a` auf den Stapel
  `struct element *ablegen(struct element *top, char a)`
- Entnehmen des zuletzt abgelegten Buchstabens vom Stapel
  `struct element *entnehmen(struct element *top, char *a)`
- Testen ob der Stapel leer ist
  `int empty(struct element *top)`

Verwenden Sie die zur Verfügung gestellte Datei `lifo_test.c` und ergänzen Sie den dort bereits vorhandenen Code um eine Implementierung der oben angegebenen Funktionen. Testen Sie Ihre Implementierung!

1) Strukturen einander zuweisen

Modernen C-Compiler, insbesondere ANSI-C, erlauben es, Strukturen des gleichen Typs einander zuzuweisen:

```
struct typName  s1;
struct typName  s2;
…
s2=s1;   /* … oder s2=s1 */
```

In der Vorlesung wurde der benutzerdefinierte Datentyp Artikel besprochen.
Die Schnittstelle des Datentyps enthält folgende Funktion:

```
void swapArtikel(Artikel *this, Artikel *other);
```

b.w.

Überlegen Sie sich, wie sich diese Funktion mit Hilfe der obigen Zuweisung einfacher gestalten lässt. Implementieren und testen Sie anschließend die neue Version.

2) *Benutzerdefinierter Datentyp I*
Gegeben sei folgendes Strukturmuster

```
typedef struct printjob {
   int pid;              /* Process id */
   char user[20];        /* username   */
   unsigned int pages;   /* page count */
} PrintJob;
```

und die folgende Schnittstelle

```
void initPrintJob(PrintJob *this, int pid, char *user,
                  unsigned int pages);
```
(Füllt die `PrintJob`-Variable `this` mit den angegebenen Werten für `pid`, `user` und `pages` .)

```
int getPid(PrintJob *this);
```
(Holt den Wert von `pid` aus der `PrintJob`-Variablen `this` .)

```
char *getUser(PrintJob *this);
```
(Holt den Wert von `user` aus der `PrintJob`-Variablen `this` .)

```
unsigned int getPages(PrintJob *this);
```
(Holt den Wert von `pages` aus der `PrintJob`-Variablen `this` .)

```
void setUser(PrintJob *this, char *user);
```
(Speichert den Wert `user` in der `PrintJob`-Variablen `this` .)

```
void setPages(PrintJob *this, unsigned int pages);
```
(Speichert den Wert `pages` in der `PrintJob`-Variablen `this` .)

```
void copyPrintJob(PrintJob *src, PrintJob *dest);
```
(Vergibt neuen Wert von `pid` und kopiert alle übrigen Attribute von `src` nach `dest` .)

```
void showPrintJob(PrintJob *this);
```
(Gibt den Inhalt der `PrintJob`-Variablen `this` aus.)

Implementieren Sie diese Schnittstelle, indem Sie analog zum Beispiel 6-7-1-Artikel (vgl. Seiten 64 und 65 im Skript) eine Header-Datei `PrintJob.h` und eine zugehörige Quelltext-Datei `PrintJob.c` erstellen!

Testen Sie Ihre Implementierung mit dem Hauptprogramm `testPrintJob.c` oder `testPrintJob-array.c`, die beide bereitgestellt werden!

Hinweis zu LINUX:

Kompilieren Sie `PrintJob.c` mit: `gcc PrintJob.c -c -o PrintJob.o`,

kompilieren Sie `testPrintJob.c` mit: `gcc testPrintJob.c -c -o testPrintJob.o`,

binden Sie die beiden Objektdateien mit: `gcc testPrintJob.o PrintJob.o`,

starten Sie `a.out` !

Hinweis zu Dev-C++ unter Windows:

Erzeugen Sie für das Projekt ein eigenes Verzeichnis und verschieben Sie die Header-Datei und die Quelltext-Datei dorthin!

Erzeugen Sie in Dev-C++ mit dem Menüpunkt `Datei|Neu` ein neues Projekt vom Typ „Console Application“ und „C-Projekt“!

Ersetzen Sie den Inhalt der neuen Datei `main.c` durch den Inhalt Ihres Hauptprogramms!

Fügen Sie mit dem Menüpunkt `Projekt|Zum Projekt hinzufügen` die Header-Datei und die Quelltext-Datei hinzu!

Speichern Sie das Projekt im neuen Projektverzeichnis!

Künftig können Sie das Projekt durch Doppelklick auf die `dev`-Datei öffnen und dann bearbeiten.

3) *Benutzerdefinierter Datentyp II*

Implementieren Sie einen Datentyp `Stack`, der nach dem LIFO-Prinzip arbeitet. Der Stack soll eine Reihe einzelner Zeichen aufnehmen.

Verwenden Sie dazu folgende Datenstruktur

```
typedef struct stack {
   int depth;                    /* Tiefe des Stacks */
   char* content;                /* Inhalt des Stacks */
   int top;                      /* Index des obersten Elements */
}Stack;
```

Implementieren Sie die Schnittstelle mit folgenden Funktionen:

`Stack *createStack(unsigned int);`
(Erzeugt eine `Stack`-Variable in der angegebenen Größe.)

`void push(Stack *, char);`
(Stellt ein Zeichen in eine `Stack`-Variable.)

`char pop(Stack *);`
(Holt ein Zeichen aus einer `Stack`-Variablen.)

`int isEmpty(Stack *);`
(Prüft, ob eine `Stack`-Variable leer ist.)

`void destroyStack(Stack *);`
(Löscht eine `Stack`-Variable vom Heap.)

Erstellen Sie eine entsprechende Header-Datei `Stack.h` und die zugehörige Quelltext-Datei `Stack.c` !

Testen Sie Ihre Implementierung mit Hilfe des bereitgestellten Hauptprogramms `testStack.c` !