PT2: Serie 1

Abgabetermin	**21.04.2018, 21:00 Uhr**
Übungstermin	*entfällt*

Implementieren Sie die statische Bibliothek `libarena.a`, die eine einfache Freispeicherverwaltung unter folgender Schnittstelle anbietet:

```
/* allocate.h */
void *allocate();
void deallocate(void *data);
```

Der verwaltete Speicherbereich sei wie folgt deklariert (Vorlage):

```
#define BLOCKSIZE 40
#define NUM_BLOCKS 1024
extern uint8_t  arena[BLOCKSIZE*NUM_BLOCKS];
extern uint16_t allocated_map[NUM_BLOCKS/16];
```

Die Funktion `allocate()` alloziert Blöcke fester Größe (`BLOCKSIZE`); die Funktion `deallocate()` gibt diese Blöcke wieder frei.

Falls alle Blöcke alloziert sind, liefert `allocate()` als Ergebnis 0. Das Feld `allocated_map` speichert mit einem Bit pro Block, welche Blöcke alloziert sind.

Abgabe

Reichen Sie Ihre Lösung in Form eines einzelen gzip-komprimierten Tarfiles ein. Dieses sollte im Wurzelverzeichnis ein Makefile haben, mit den Zielen `libarena.a`, `testapp` (basierend auf `testapp.c`) und `all`.

Gehen Sie davon aus, dass `testapp.c` auch gegen andere Bibliotheken gelinkt werden können soll. Für Ihr äußeres Makefile soll die Variable `LIBARENA` ein Verzeichnis angeben, in dem sich `libarena.a` und `allocate.h` befinden.

```
/
|--testapp.c
|--Makefile
|--libarena/
   |--allocate.h
   |--allocate.c
   |--arena.h
   |--arena.c
   |--Makefile
```

Zusatzaufgabe

Erweitern Sie Ihre Speicherverwaltung auf mehrere Arena-Blöcke. Verwenden Sie `malloc()` / `free()` zum Anlegen der Arenas. Implementieren Sie dazu die Funktionen

```
void *newArena(int blocksize, int numblocks);
void freeArena(void *arena);
```

Die Funktionen

```
void *allocateEx(void *arena);
void deallocateEx(void *arena, void *data);
```

sollen wie `allocate()` und `deallocate()` funktionieren, und zusätzlich einen Verweis auf die zu verwendene Arena enthalten. Zur Verwaltung der verschiedenen Arenas wird eine einfach-verkettete Liste empfohlen.