SYSTEMPROGRAMMIERUNG: 04a. Aufgabenblatt

WS 2013/2014
Prof. Dr.-Ing. C. Gips, Dipl.-Inf. Birgit C. George

FH Bielefeld
University of
Applied Sciences

Speicherverwaltung selbst gemacht (Teil 1)

Ausgabe:	**05.11.2013**
Abgabe:	**KW46 (12./15.11.2013)**
Punkte:	**0+4**

Teil 1/2

Auf diesem Blatt soll eine einfache Freispeicherverwaltung wie in der Vorlesung besprochen implementiert werden.

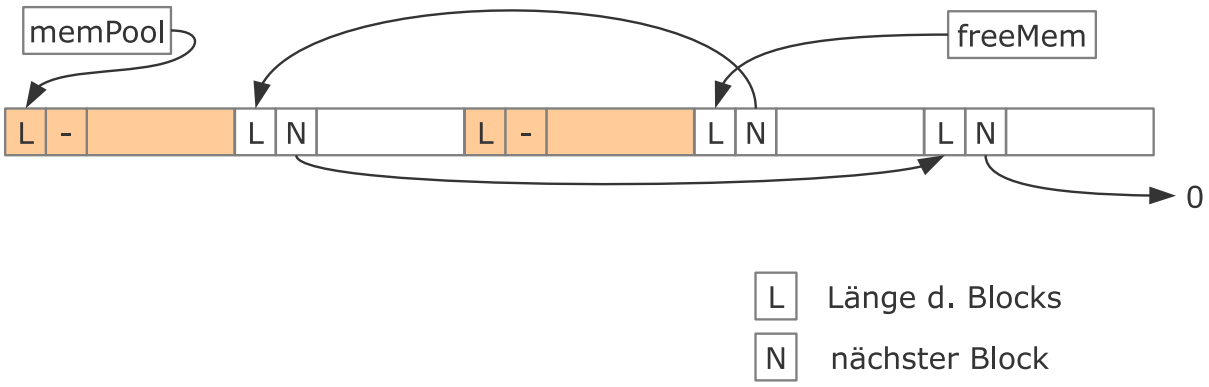
- Der Heap soll durch ein ausreichend großes Byte-Array `memPool` simuliert werden:

```
1  char memPool[MEM_POOL_SIZE];
```

- Eine Vergrößerung/Verkleinerung des simulierten Heaps während der Laufzeit des Programmes ist nicht vorgesehen.
- Die freien Blöcke sollen über eine einfach verlinkte Liste verwaltet werden. Dazu enthält jeder Block im simulierten Heap am Anfang eine Verwaltungsstruktur, die die Gesamtlänge des Blocks (in Bytes) und einen Zeiger auf den Anfang des nächsten freien Blocks enhält:

```
1  struct MemBlock {
2      size_t size;
3      struct MemBlock *next;
4  };
```

- Der `next`-Pointer des letzten Blocks ist ein Null-Pointer.
- Der Pointer `freeMem` (Typ: `struct MemBlock*`) zeigt auf den Anfang der Freispeicherliste.
- Belegte Blöcke sollen nicht in der Freispeicherliste verwaltet werden, sie behalten aber ihre Verwaltungsstruktur am Anfang des Blockes. Um belegte Blöcke extra kenntlich zu machen, soll der `next`-Pointer belegter Blöcke den „magischen“ Integerwert `0xdeaddead` enthalten.



Achten Sie darauf, *alle* von Ihnen belegten Ressourcen auf dem Heap-Speicher wieder freizugeben, sobald diese nicht mehr benötigt werden!

Schreiben Sie eine `main()`-Funktion, in der die einzelnen Funktionen aufgerufen werden (können).

1. Aufgabe: Strukturen und Variablen — 1 Punkt (Bonus)

- Definieren Sie zunächst geeignete Konstanten:
  1. Größe des simulierten Heaps: 16384 Bytes
  2. „magischer“ Integerwert `0xdeaddead`
- Legen Sie den simulierten Heap `memPool` und den Pointer `freeMem` als globale Variablen an.
- Definieren Sie nun die benötigten Datentypen und Strukturen für die Verwaltungsstruktur `struct MemBlock`.

Ziel: Umgang mit Strukturen und Arrays

2. Aufgabe: Initialisierung — 1 Punkt (Bonus)

Schreiben Sie eine parameterlose Funktion `void initHeap(void)`, die den simulierten Heap initialisiert und den Pointer `freeMem` auf den Beginn der Freispeicherliste zeigen lässt. Gestalten Sie die Funktion so, daß sie nur beim ersten Aufruf den Heap initialisiert. Bei allen folgenden Aufrufen soll sie nichts tun. Dies können Sie ohne weitere globale Variablen erreichen!

Hinweis: Der initiale (leere) Heap wird als ein einziger freier Block betrachtet.

Ziel: Strukturen und Pointerarithmetik

3. Aufgabe: Adressumrechnung — 1 Punkt (Bonus)

Im Heap werden alle Adressen relativ zur Startadresse vom `char`-Array benötigt. Schreiben Sie sich zwei Funktionen, mit denen Sie eine relative Adresse im virtuellen Heap in eine absolute Adresse und umgekehrt umrechnen können.

Ziel: Adressumrechnung

4. Aufgabe: Ausgabe der freien Blöcke — 1 Punkt (Bonus)

Implementieren Sie eine Funktion `void printBlock(struct MemBlock *p)` für die Ausgabe der Verwaltungsinformationen eines Speicherblocks. Diese muss für einen Block `p` folgende Daten übersichtlich ausgeben:

- Startadresse der Verwaltungsstruktur
- Größe der Verwaltungsstruktur (in Bytes)
- Startadresse des nutzbaren Speichers
- Größe des nutzbaren Speichers (in Bytes)
- Gesamtgröße des Blocks (Größe des nutzbaren Speichers plus Größe Verwaltungsstruktur in Bytes)
- Wert des `next`-Pointers
- Hinweis, ob der Block frei oder belegt ist

Verwenden Sie Dezimalzahlen für die Ausgabe der Adressen und Größen! Geben Sie die Adressen jeweils zusätzlich relativ zur Startadresse aus.

Implementieren Sie die Funktion `void printFreeBlocks(void)`. Diese soll **alle** freien Blöcke in der Reihenfolge, in der sie in der Freispeicherliste stehen, ausgeben. Nutzen Sie dazu die Funktion `void printBlock(struct MemBlock *p)`.

Ziel: Iteration durch verkettete Liste, formatierte Ausgabe