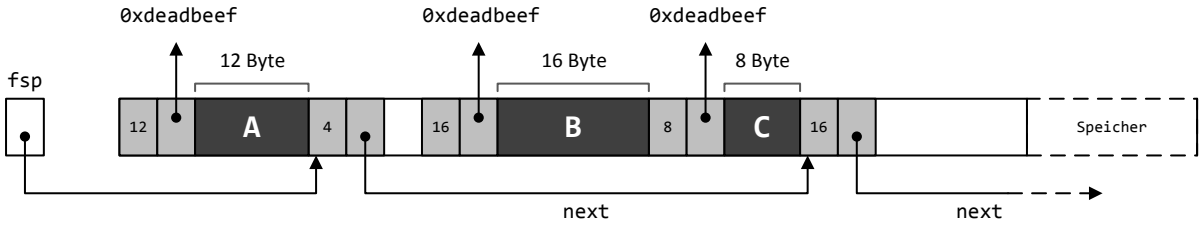
Aufgabenblatt #R5 zur Rechnerübung Betriebssysteme

In dieser Aufgabe soll eine einfache Freispeicherverwaltung implementiert werden, welche die Funktionen **malloc(3)**, **calloc(3)**, **realloc(3)** und **free(3)** aus der Standard-C-Bibliothek ersetzt. Verwaltet werden soll hier ein statisch allokierter Speicher von 1024 Byte (=1 KiB). Zur Verwaltung der einzelnen Speicherbereiche ist die Verwaltungsstruktur `memblock` aus der Übung vorgegeben. Sie besitzt folgende Member:

- `size_t size`: Gibt die Größe des Speicherbereichs an, der nach der Verwaltungsstruktur folgt.
- `struct memblock *next`: Ist der nachfolgende Speicher frei, so zeigt dieser Zeiger entweder auf die Verwaltungsstruktur des nächsten freien Speicherbereichs oder ins Nichts (`NULL`)
  Sollte jedoch der nachfolgende Speicherbereich belegt sein so steht in diesem Zeiger die magische Ziffer `0xdeadbeef`.

Der Freie Speicher wird damit als einfache, verkettete Liste verwaltet. Ein Zeiger, hier *free space pointer* (`fsp`) genannt, zeigt auf den Anfang der Liste.



Vorgegeben sind die Dateien *mymem.c*, *mymem.h*, ein Testfall in *test.c* sowie das gewünschte Ergebnis des Testfalls in *mempicture.txt*. Ihre Aufgabe ist es in der *mymem.c* die nachfolgenden Funktionen zu implementieren. Bei korrekter Implementierung sollte ihr Ergebnis mit dem in *mempicture.txt* übereinstimmen.

5.1 `void *myMalloc(size_t size)`

Die Funktion `malloc()` bekommt als Argument die Größe `size` des zu allokierenden Speicherbereichs. Als Rückgabewert gibt sie im Erfolgsfall einen Zeiger auf den Anfang des allokierten Speicherbereichs.
Implementieren Sie diese Funktionalität in `myMalloc()`. Gehen sie dabei wie folgt vor:

0. Wenn der `fsp` noch auf `null` gesetzt ist, initialisieren sie den Speicher indem Sie am Anfang eine Verwaltungseinheit anlegen. Anschließend sollte `fsp` auf diese Verwaltungseinheit zeigen.
1. Suchen Sie einen passenden freien Speicherbereich für die gewünschte Datenmenge
2. Reservieren Sie den Speicherbereich indem Sie die entsprechenden Informationen in die Verwaltungsstruktur eintragen.
3. Legen Sie hinter dem reservierten Speicherbereichs eine neue Verwaltungsstruktur an
4. verschieben sie den `fsp`

5.2 `void myFree(void* ptr)`

Die Funktion `free()` gibt den Speicher frei der von `malloc()` allokiert wurde. Ein ähnliches verhalten sollen Sie nun für myFree() implementieren.
Die Funktion `myFree()` gibt einen von `myMalloc()` allokierten und über den Zeiger `ptr` identifizierten Speicherbereich frei. Hierfür wird in der zugehörigen Verwaltungsstruktur die *Magic Number* entfernt und der nun freigewordene Speicherbereich wird in die verkettete Liste eingehängt. Der Einfachheit halber soll freigewordener Speicher nicht mit eventuellen benachbarten freien Bereichen verschmolzen werden. Gehen sie zur bei der realisierung dieser Aufgabe wie folgt vor:

1. Berechnen Sie zuerst die Adresse der für den Speicherbereich verantwortlichen Verwaltungsstruktur
2. Prüfen Sie ob der Speicher wirklich belegt ist.
3. Setzen Sie den `fsp` um und hängen Sie die Verwaltungsstruktur an korrekter Stelle in die verkettete Liste ein. Dabei müssen die `next`-Zeiger umliegender Verwaltungsstrukturen eventuell aktualisiert werden.

5.3 Optional: `void *myCalloc(size_t count, size_t size)`

Die Funktion `calloc()` ist der Funktion `malloc()` sehr ähnlich. Wie bei `malloc()` wird auch bei `calloc()` eine zusammenhängende Menge Speicher allokiert und ein Zeiger auf den Anfang dieses Speicherbereichs zurückgegeben. Der Unterschied liegt darin, dass bei `calloc()` die Größe des zu reservierenden Speichers $count * size$ beträgt und der Speicherbereich vollständig mit 0 initialisiert wird.
Implementieren Sie mit `myCalloc()` die Funktionalität von `calloc()`. Verwenden sie bei ihrer Implementierung intern die myMalloc-Funktion um den Speicher zu allokieren.

5.4 Optional: `void *myRealloc(void *ptr, size_t size)`

Die realloc-Funktion versucht den Speicherbereich der über den Zeiger `ptr` identifiziert wird auf eine neue Größe `size` anzupassen. Sie sollen mit `myRealloc()` eine vereinfachte Version implementieren die intern `myMalloc()`, `memcopy()`, und `myFree()` verwendet:

1. Fordern sie mittels `myMalloc()` einen neuen Speicherbereich von der Größe `size` an.
2. Kopieren sie den Inhalt vom gegebenen Speicherbereich (s. `ptr`) in den neu allokierterten Speicherbereich.
3. Geben sie den gegebenen Speicherbereich (s. `ptr`) mittels `myFree()` wieder frei.

Als Rückgabewert sollte myRealloc einen Zeiger auf den Anfang des neu allokierten Speicherbereichs zurückgeben.

Abgabe: bis 11.01.2013