Aufgabe 5: Speicherverwaltung

*Als Vorbereitung zu dieser Aufgabe schauen Sie sich bitte die Folien aus der großen Übung an. Es ist dabei wichtig, dass Sie den Speicherverwaltungsalgorithmus soweit verstanden haben, dass Sie das Speicher-Layout nach einigen Iterationen von **malloc(3)** und **free(3)**, inklusive korrekter Zeigerverlinkung, auf einem Blatt Papier zeichnen können.*

In dieser Aufgabe sollen Sie eine einfache Variante von **malloc(3)** und **free(3)** implementieren und damit die entsprechenden Funktionen aus der Standard-C-Bibliothek ersetzen. Verwaltet werden soll hier ein Speicher von 4096 Byte (=4 KiB), der via **mmap(3)** alloziert wird. Zur Verwaltung der einzelnen Speicherbereiche ist die Verwaltungsstruktur `memory_block_t` aus der Übung vorgegeben. Sie besitzt folgende Membervariablen:

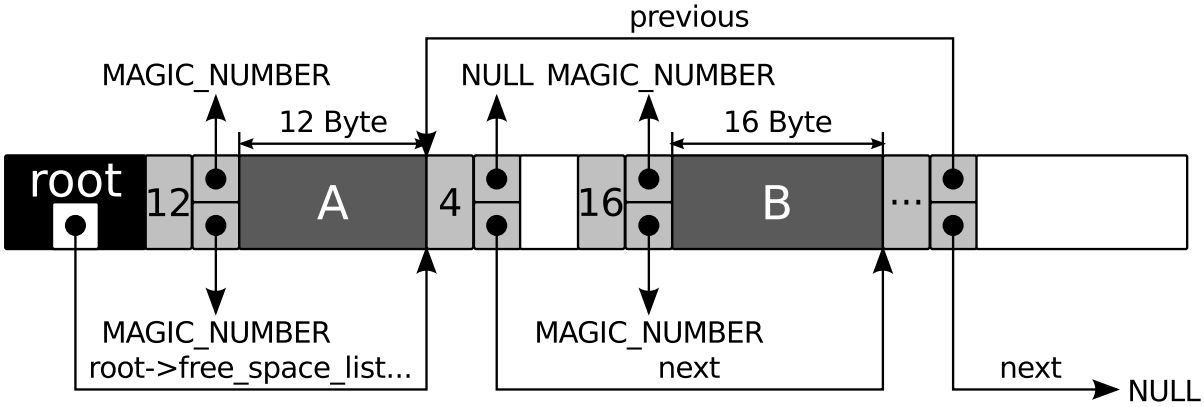
- `size`: gibt die Größe des Speicherbereichs in Byte an, der nach der Verwaltungsstruktur folgt
- `next`: zeigt auf die Verwaltungsstruktur des nächsten freien Speicherbereichs oder auf `(NULL)`.
- `previous`: zeigt auf die Verwaltungsstruktur des vorhergehende freien Speicherbereichs oder auf `(NULL)`.

Sollte der zu verwaltende Speicherbereich belegt sein, so steht in Zeigern `next` und `previous` eine `MAGIC_NUMBER` (`0xC0CAC01AADD511FE`).

Der freie Speicher wird somit als doppelt verkettete Liste verwaltet. Damit überhaupt Speicher für die Verwaltung bereitsteht, soll `mmap` genutzt werden. Durch diesen Systemaufruf wird das Betriebssystem angewiesen, eine Seite direkt in den Adressraum des Programms abzubilden. Für eine einfache Handhabung soll eine Datei in den Adressraum des Programms abgebildet werden.

Für die Verwaltung der Datei und der Seite gibt es neben der Verwaltungsstruktur `memory_block_t` noch `chunk_t`. In diese Struktur werden alle Informationen bezüglich der abgebildeten Datei und des Einstiegspunktes zum ersten freien Speicherbereich abgespeichert. Die Membervariablen dieses Struktur sind die folgenden:

- `file`: dieser File-Descriptor verweist auf die Datei, welche mittels `open` angelegt und geöffnet wurde
- `size`: diese Membervariable enthält die Größe des zu verwaltenden Speichers
- `free_space_list_entry_point`: dieser Zeiger verweist auf das erste Element der Freispeicherliste



Vorgegeben sind die Dateien `include/colorCodes.h`, `include/mymalloc.h` sowie alle Tests in `src/main.c`. Sie finden zudem im Ordner `ressources` die Ergebnisse der einzelnen Implementierungsstufen in Form von Speicherabbildungen in Hexadezimaldarstellung. Ihre Aufgabe ist es, in der `src/mymalloc.c` die nachfolgenden Funktionen zu implementieren.

Testen: Zum Testen Ihrer Implementierung können Sie das mitgelieferte Skript `run_compare.bash` mit einem von drei folgenden Parameter über das Terminal aufrufen: `-no-free, -no-merge, -final`
Dieses Skript startet für Sie die Testapplikation `./mymalloc` und erzeugt anschließend mit **od(1)** ein gut lesbares Speicherabbild in Hexadezimaldarstellung. Das erzeugte Speicherabbild wird dann via **meld(1)** mit der zu den Tests passenden Musterlösung in `memory_map_*.hex` verglichen. Alternativ können Sie auch `./mymalloc` mit den gleichen Parametern direkt über das Terminal aufrufen. Diese Tests werden auch in der GitLab CI durchgeführt und können dort wie gewohnt eingesehen werden.

5.1 Implementierung von `void* my_malloc(size_t size)` (12 Punkte)

Als vorbereitenden Schritt gilt es, den Speicher für die `my_malloc` Funktionalität zu reservieren und für die spätere Verwendung vorzubereiten. Implementieren Sie zu diesem Zweck die vorgegebene Funktion `open_file()`. Hierbei müssen Sie eine Datei (Dateiname: `FILE_NAME`) mittels **open(2)** öffnen bzw. erstellen, wenn diese nicht existiert. Beachten Sie bei `open` den *mode*-Parameter. Dieser sollte so gewählt werden, dass nur der Besitzer der erzeugten Datei Lese- und Schreibrechte hat. Anschließend müssen Sie in der Datei Speicher in der Größe einer Seite mittels **posix_fallocate(3)** reservieren.

Bei der Implementierung von `my_malloc` sollten Sie wie folgt vorgehen:

1. Wenn die `root`-Struktur noch auf `NULL` gesetzt ist, müssen Sie Speicher via `mmap` anlegen.
   - Öffnen Sie zuerst mittels `open_file()` eine Datei.
   - Bilden Sie die geöffnete Datei anschließend via `mmap` in den Adressraum Ihres Programms ab. Damit der Speicher immer an einer festen, leicht lesbaren Adresse liegt, sollten Sie `mmap` eine Start-Adresse (`START_ADDRESS`) übergeben und `mmap`, mittels dem Flag `MAP_FIXED`, anweisen, diese Adresse zwingend zu nutzen. Außerdem benötigen Sie das Flag `MAP_SHARED`. Schauen Sie in die Man-Page von `mmap`, um die Bedeutung des Flags zu erfahren. Achten Sie bitte auf eine korrekte Fehlerbehandlung: wenn `mmap`, `open` oder `posix_fallocate` fehlschlagen sollte, muss der Prozess beendet werden.
   - Der nun reservierte Speicher wird der `root`-Struktur zugewiesen und die Membervariablen initialisiert.
   - Direkt nach dem Wurzelknoten legen Sie den ersten `memory_block_t` an und lassen den `free_space_list_entry_point` der Wurzelstruktur auf diesen zeigen.
2. Berechnen Sie die zu reservierende Speichermenge als ein aufgerundetes Vielfaches von `memory_block_t`.
3. Suchen Sie den ersten, passenden freien Speicherbereich für die gewünschte Datenmenge. Hierbei sind folgende Fälle zu beachten:
   - 1. Fall: Ein passender freier Speicherbereich konnte gefunden werden.
   - 2. Fall: Es existiert kein zusammenhängender freier Speicherbereich, der für die angeforderte Datenmenge groß genug ist → setzen Sie `errno` auf `ENOMEM` und geben Sie `NULL` zurück.
4. Reservieren Sie den gefundenen Speicherbereich, indem Sie die entsprechenden Informationen (`MAGIC_NUMBER` sowie die neue Größe) in die dazugehörige Verwaltungsstruktur eintragen.
5. Falls hinter der zu reservierenden Datenmenge noch freier Speicher existiert, legen Sie hierfür eine neue Verwaltungsstruktur an und versehen Sie diese mit passenden Werten. Hierbei ist folgendes zu beachten:
   - Der `next`-Zeiger muss auf die nächste Verwaltungsstruktur zeigen und `size` muss korrekt berechnet werden; sollte kein folgender freier Speicher vorhanden sein, zeigt der `next`-Zeiger auf `NULL`.
   - Der `previous`-Zeiger muss auf die vorherige Verwaltungsstruktur zeigen; sollte kein vorheriger freier Speicher vorhanden sein, zeigt dieser auf `NULL`.
   - Der Zeiger eines weiter vorne liegenden freien Speicherbereichs muss auf den neuen freien Speicherbereich zeigen, bzw. auf seine `memory_block_t`-Verwaltungsstruktur.
   - Der `free_space_list_entry_point` muss eventuell verschoben werden.

Überprüfen Sie Ihre Implementierung in den GitLab CI Tests oder testen Sie sie mit dem mitgelieferten Skript `run_compare.bash` bzw. direkt mit `mymalloc` jeweils unter Verwendung des Parameters `-no-free`.

5.2 Einfache Implementierung von `void my_free(void*)` (11 Punkte)

Die Funktion `my_free()` gibt einen von `my_malloc()` allozierten und über den Zeiger `memory_location` identifizierten Speicherbereich frei. Hierfür wird in der zugehörigen Verwaltungsstruktur die `MAGIC_NUMBER` entfernt und der nun freigewordene Speicherbereich wird in die verkettete Liste eingehängt. Freigewordener Speicher soll mit eventuellen benachbarten freien Bereichen verschmolzen werden. Gehen Sie bei der Realisierung dieser Aufgabe wie folgt vor:

1. Berechnen Sie ausgehend vom übergebenen Zeiger die Adresse der für den Speicherbereich verantwortlichen Verwaltungsstruktur.
2. Überprüfen Sie, ob es sich wirklich um eine Verwaltungsstruktur eines belegten Speicherbereichs handelt.
   - Belegter Speicher ist durch die `MAGIC_NUMBER` in der `next`- und `previous`-Membervariable der `memory_block_t`-Verwaltungsstruktur zu finden.
   - Sollte dies nicht der Fall sein oder sollte der Wurzelknoten nicht existieren, beenden Sie die Funktion mit einer Fehlermeldung.
   - Sollte der übergebene Zeiger nicht gültig sein, beenden Sie die Funktion ohne eine Fehlermeldung (analog zur `GNU libc`-Implementierung).
3. Setzen Sie den `free_space_list_entry_point` um und hängen Sie die Verwaltungsstruktur an korrekter Stelle in die verkettete Liste ein. Beachten Sie hierbei folgende Fälle:
   - 1. Fall: Der gefundene `memory_block_t` liegt vor dem aktuell ersten freien Bereich (gekennzeichnet durch `free_space_list_entry_point`).
   - 2. Fall: Der gefundene `memory_block_t` liegt liegt zwischen zwei freien Bereichen.
   - 3. Fall: Der gefundene `memory_block_t` liegt hinter dem letzten freien Speicherbereich.
   - 4. Fall: Es kann kein Bezug zu vorherigen / nachfolgenden Blöcken gefunden werden - hier sollten Sie die Funktion mit einer Fehlermeldung beenden.

Bei korrekter Implementierung sollte Ihr Ergebnis mit dem in *memory_map_no_merge.bin* übereinstimmen (Start des Programms mit dem Parameter `-no-merge`).

Überprüfen Sie Ihre Implementierung in den GitLab CI Tests oder testen Sie sie mit dem mitgelieferte Skript `run_compare.bash` bzw. direkt mit `mymalloc` jeweils unter Verwendung des Parameters `-no-merge`.

5.3 Erweiterte Implementierung von `void my_free(void*)` (6 Punkte)

Sie können nun mit `my_free` zwar Speicher freigeben, doch werden freie, benachbarte Speicherbereiche nicht zusammengeführt. Als Folge können mit der Zeit immer kleinere Speicherfragmente alloziert werden, bis zu dem Punkt, wo nur noch Allokationen in der Größe einzelner Memory-Blöcke angelegt werden können. Dieses sollen Sie nun ändern, indem bei jeder Speicherfreigabe benachbarte freie Speicherbereiche gesucht und verschmolzen werden.

Beachten Sie hierbei die folgenden drei Fälle:

- 1. Fall: Es können keine freien Blöcke miteinander verschmolzen werden.
- 2. Fall: Nur zwei Blöcke können miteinander verschmolzen werden (vorheriger bzw. nachfolgender Block).
- 3. Fall: Drei Blöcke können miteinander verschmolzen werden.

Bei korrekter Implementierung sollte Ihr Ergebnis mit dem in *memory_map_final.bin* übereinstimmen (Start des Programms ohne Parameter).

Überprüfen Sie Ihre Implementierung in den GitLab CI Tests oder testen Sie sie mit dem mitgelieferten Skript `run_compare.bash` bzw. direkt mit `mymalloc` jeweils unter Verwendung des Parameters `-final`.

Abgabe bis 14.12.2018, 23:59 Uhr.

Ihre Lösungen sollen bis zur Deadline im GitLab des IBR hochgeladen sein.
Einhaltung der Coding Guidelines: 3 Punkte
Zu erzielende Minimalpunktzahl: 16 Punkte