CS3026 - Assessment 1 Memory Management

File Structure

```
──cs3026_assessment1_AndrejSzalma
    ├── CGS_A5_A1
    │   ├── Makefile
    │   ├── mymemory.c
    │   ├── mymemory.h
    │   └── shell.c
    ├── CGS_B3_B1
    │   ├── Makefile
    │   ├── mymemory.c
    │   ├── mymemory.h
    │   └── shell.c
    ├── CGS_C3_C1
    │   ├── Makefile
    │   ├── mymemory.c
    │   ├── mymemory.h
    │   └── shell.c
    └── CGS_D3_D1
        ├── Makefile
        ├── mymemory.c
        ├── mymemory.h
        └── shell.c
```

Every directory represents a group of source files based on the assesment instructions. Starting from the simplest functions implemented in CGS_D3_D1 the files are being expanded with new functions and their implementations up till CGS_A5_A1.

Usage & Specifications

This series of programs were developed to work with the `x86_64-linux-gnu` compiler. The `gcc version 7.5.0` was used to compile the files during the development.

Every folder contains a makefile, which can be run by the `make` command in an unix/linux terminal. This command will compile the source code and clean up after compilation. The resulting `shell` file is an executable which can be run by typing `./shell`.

CGS_D3_D1

The aim of this version of the program was to develop three functions: - `initialize();` initializes the memory array with '\0's and creates an unallocated segment descriptor - `printmemory();` prints the memory array in a nice format which can be used to debug/inspect the state of the memory. The format of a line printed by the function is `[index of first Byte in line] [10 Bytes in hex] | [same 10 bytes in ACSII]` - `printsegmenttable();` prints the linked list holding all the segment descriptors in a list format.

CGS_C3_C1

The aim of this version of the program was to develop own version of the `malloc()` function: - `void * mymalloc( size_t );` takes a number of Bytes and allocates a free spacing in memory according to that. Returns the pointer to the start address in the memory.

CGS_B3_B1

The aim of this version of the program was to develp own version of the `free()` function: - `void myfree( *ptr );` takes a pointer to the start address in the memory where some memory was previously allocated and frees up this space, for further allocation.

CGS_A5_A1

The aim of this version of the program was to develop a function to defragment the memory: - `void mydefrag ();` Takes no parameters as it defragments the whole memory, merging all the free segment descriptors and moving all the allocated ones (including their data saved in memory) to the start of the memory. Even though this would have no affect on the outcome of this function, the order of the allocated memory is retained.

Andrej Szalma - 51983729 - 01/11/2021