```
/**
 * Assignment 1: ++Malloc README
 *
 * @author Taran Suresh ts875
 * @author Abhishek Naikoti an643
 */
```

--- Program Description ---

This program implements the popular *malloc()* and *free()* library calls, but in a rather refined manner that prevents the user from doing Bad Things. A large array of size 4096 (*myblock*) was used to simulate main memory, wherein metadata and userdata are stored. When a valid call to *malloc()* is made, a pointer to the userdata is returned, and when a valid call to *free()* is made, the previously allocated userdata is freed. Invalid calls to both *malloc()* and *free()* are handled gracefully with errors, and allow the user to continue using the library calls.

--- Compile & Run on the Command Line ---

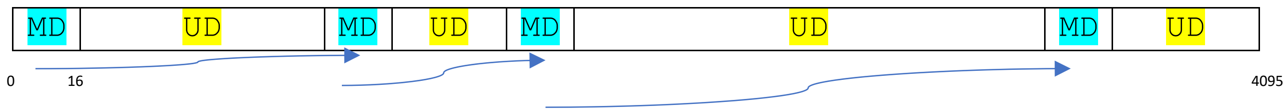
1. Navigate into the project directory
2. Input the following command to compile: *make*
3. Input the following command to run: *./memgrind*
4. Note: this program does not accept any user arguments, all calls to *malloc()* and *free()* should be done through *memgrind.c*

--- Implementation ---

This program has a couple of main features that allow for *malloc()* and *free()* to operate efficiently and gracefully. These include the *metadata, userdata,* and *coalescing*. The *metadata* was created using a singly linked list, with each node being a container for useful information regarding the respective *userdata*. These components include *blockstatus*, *blocklength*, and *next pointer.* The *metadata* always precedes the *userdata*, and currently takes *16 bytes* each. The *userdata* is simply defined as the space between two addresses, which are all computed with just the information stored in the respective *metadata* container, as well as the address at which the current *metadata* is located. Lastly, *coalescing* is a useful feature that improves the chances of the user being able to receive a pointer to some address in *myblock* where they can access their requested data.

Coalescing simply put is just combining adjacent memory blocks that are free into one big block, therefore reducing fragmentation throughout *myblock*. This is done by a traversal through the *metadata* linked list, and a deletion of a *metadata* node if *coalescing* is able to successfully occur. This also provides the user with an extra *16 bytes* (*metadata* size), as there is now one less *metadata* node in *myblock*.

--- MetaData Visual ---



--- Time & Space Complexity ---

The main data structure being used here is a linked list. Both functions *malloc()* and *free()* utilize this linked list, whose head pointer is located at the 0^{th} index of *myblock*. Traversing a linked list in the worst case is O(n) where n = number of *metadata* nodes = number of *userdata* blocks.

--- Cool Features ---

There are two extra functions *printMemory()* and *printMetaData()* available to the user if a visual of *myblock* is desired after calling *malloc()* and *free()*. Calling *printMemory()* is not always the easiest way to see what is going on as it is tough to know what type of data the user has decided to store in their allocated block, but calling *printMetaData()* provides a high level visual of the linked list which can provide useful information for debugging purposes. This information includes the length of *userdata*, free/used status of that block, as well as addresses of current *metadata* block and *next metadata* block. These addresses have also been converted into decimal form, rather than the usual hexadecimal form for easier readability (0 – 4095).

--- Testing ---

Refer to memgrind.c and testcases.txt to see more about the test cases regarding the *malloc()* and *free()* library calls.

--- Project Directory ---

- readme.pdf
- testcases.txt

- mymalloc.h
- mymalloc.c
- memgrind.c
- Makefile