CS214: Systems Programming
Assignment 1 ++Malloc
Gazal Arora and Whitney Alderman

This assignment designs and implements a different version of malloc() and free() library functions for dynamic memory allocation.
The functions are called mymalloc() and myfree() but can be used as malloc() and free() by using the following macros in the mymalloc.h file and including mymalloc.h in your own version of memgrind.c

```
#define malloc( x ) mymalloc( x, __FILE__, __LINE__ )
#define free( x ) myfree( x, __FILE__, __LINE__ )
```

The assignment contains the following files:
mymalloc.c: C file with implementation of mymalloc and myfree
memgrind.c: C file with thorough testing of mymalloc and myfree using five different workloads
mymalloc.h: Header file containing all library calls, definitions of macros and structs
Makefile: File that compiles mymalloc.c and memgrind.c into executables that are linked to each other for running
readme.pdf: Familiarizing user with the project
testcases.txt: A file explaining workload D and E from memgrind.c

Compilation: To compile mymalloc.c and memgrind.c simultaneously, please write the following command in the terminal on ilab machine
make memgrind

Run: To run the executables, please type the following after compilation in the terminal on ilab machine
./memgrind

<u>mymalloc(int bytes) Implementation</u>
We implement dynamic allocation in an array of size 4096 bytes. We have struct of size 16 bytes to maintain the metadata of each allocated or free block.
The metadata struct is made up of two integers, one stores the size of the block, second stores a 0 if block is not free and 1 if block is free. It also consists of a struct metadata pointer pointing to the next block.
The memory array is embedded with a linked list of these metadata structs. The user can call on malloc(x bytes) or free(pointer) to allocate memory and get a pointer to that

memory or to free a memory space using its pointer, respectively.

When malloc is called for the first time, the memory block array is initialized with a single metadata.

After this step, we traverse the array using metadata pointers to go to the next block of memory until we find a free block. If the size of free block is large enough for the memory asked by the user and for a new metadata pointing to the next free block then this free block is broken or split into two blocks, first one pointing to user's allocated bytes and next one pointing to the next free block. If this free block is not large enough to perform a split, then the user is returned a pointer pointing to a block with either exactly the size requested, or slightly more bytes than requested. When user calls malloc, we return the pointer to the byte occurring right after the first available(large enough and free) metadata. Maximum bytes that a user can allocate in a single malloc call is 4080

Malloc Error Handling:
Malloc(0) is not allowed and returns Error message "ERROR: (File: Line:) Cannot return a pointer to a 0 byte block"

If space requested is larger than available free bytes then "ERROR: (File: Line:) Memory is not large enough for request" Error is returned.

Upon error, name of the file & line number that caused the error are printed.

myfree(void *ptr) Implementation:
When free is called, we check if the memory block array is initialized with a metadata that points to the beginning of the array or not.

After this step, we traverse the array using metadata pointers to check the byte immediately following the metadata block. The addresses are compared to see if we find the user's requested pointer to free in this memory block. Once we find the user's pointer in the memory block, we free it by setting the isfree field to 1 (true). In this step, we also check if adjacent memory blocks are free. If they are free then adjacent free blocks are combined into one free block with one metadata pointing to it so that we have more bytes available for allocation as the space taken up by the adjacent metadata's can

now be used. If we don’t find the user’s pointer in the memory array, we return an error message.

Free Error Handling:
Freeing not allowed if the memory array is uninitialized. In case of an uninitialized memory array, free returns the following error message: “ERROR: (File: Line:) No such pointer to free”

Freeing a null pointer or freeing the same pointer twice is not allowed, it will return the following error message:
“ERROR: (File: Line:) Cannot Free a NULL pointer”

Freeing pointers not allocated by malloc is not allowed. It returns the following error message:
“ERROR: (File: Line:) No such pointer found to free”

Free returns the following error message if the user’s pointer is not found in the memory array.
“ERROR: (File: Line:) No such pointer found to free”

A snapshot of the linked list of metadata’s embedded in the memory array can be seen by calling the printlinkedlist() function in your own version of memgrind.c

An example linked list on malloc(4000) call looks like the following snapshot:

```
META DATA LINKED LIST:
metadata at myblock[0:15]
[ size: 4000  isfree: 0 ]
     |
     V
metadata at myblock[4016:4031]
[ size: 64  isfree: 1 ]
     |
     V
     NULL
```