Asst1: Malloc++

1. Purpose

The purpose of this assignment is to create a more user-friendly version of the malloc and free functions.The malloc and free functions will be renamed using the following macros:

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

The __FILE__ and __LINE__ arguments passed to mymalloc and myfree will provide the file name and line number of the statement that caused an error while calling malloc or free. The user-friendly version returns a proper understandable error message instead of the vague error that usually appears: segmentation fault.

Malloc(size_t size) will return a pointer to a block of memory whose size is at least the requested size. These blocks of memory will come from a 4096-bytes array defined in mymalloc.h

```
static char myblock[4096];
```

Free(void *) will allow the user to tell the system that the user is done with the dynamically allocated memory and it can be freed for other use.

2. Some common errors that occurs while using the traditional malloc and free

Some common errors that might result in segmentation fault while trying to free a dynamically allocated memory:

- Freeing an already free memory
- Freeing a non-pointer variable
- Freeing memory that was not malloced
- Freeing a null pointer

All of these errors will be detected and handled efficiently in myfree and an appropriate error message will be displayed to the user.

Some common errors that might occur while using malloc:

- Mallocing null value
- Mallocing some number of bytes after the memory is already full
- Mallocing some number of bytes that exceeds the available memory

All of these errors will be detected and handled efficiently in mymalloc and an appropriate error message will be displayed to the user.

3. Working of mymalloc

The first call to malloc will call the initialize function which initializes our virtual memory. What the intialize() function does is that it constructs a metadata block which points to the data block which is of the size 4096 - sizeof(struct metablock). In our case, the size of the metadata block is 24 bytes. Metablock is a struct with 3 variables in it: size_t size, int free, and struct metablock next.
The next time malloc is called, malloc will search through every metablock and find the first free block whose size is greater than or equal to the number of bytes that needs to be malloced. Once such a block is found, the working is as follows:

- If the block is first or last block:
    - If the size of the block is exactly the same as the number of bytes needed to malloced, the metablock of that block sets free to 0 indicating that the block is occupied and a pointer one more than the metablock is returned.
    - If the size of the block is more than what is required, a function called split is called which results in the block occupying the space needed and converting the rest of the space into another block itself. Then the pointer to the data block is returned.
- Else
    - If the size of the block is exactly the same as the number of bytes that need to be malloced, the metablock of that block sets free to 0 indicating that the block is occupied and a pointer one more than the metablock is returned.
    - If the size of block is more than required bytes + sizeof metablock, split is called. Then the pointer to the data block, which was splitted is returned. Address of the data block is metablock + 1.

If such a block is not found, an error message is displayed which tells the user that there is no room for mallocing more bytes.

4. Working of myfree

Firstly, the function returns an error message if the argument passed to free is NULL. It also returns an error message if the head of the memoryList, a linked list representing the main mermory, is NULL because it means no data was malloced.

It then checks if the pointer passed as an argument is in bounds, that is the pointer lies between our myblock and myblock+4096 area. If it is in bounds, the function loops through our memorylist and compares the passed pointer with the datablock associated with each metablock in memorylist.

- If it matches, the memory is successfully freed
    - If the memory was freed, a function called merge is called at the end of our myfree function. The merge function merges any consecutive free blocks of memory to make a bigger free block of memory.
- Else an appropriate error message is displayed

5. Brief understanding of split()

If the block that needs to be split has a size that is greater than required bytes + 24 (which is sizeof metablock), then the size of the block is set to required bytes. And the rest of the space is allocated to a new block that follows after the block that split.

Else, it means that the extra space is even less than the amount of space the metablock requires, so the block is not split.

6. Brief understanding of merge()

If the call to myfree results in memorylist having only one block, the merge function sets the size of the datablock to 4096 - sizeof(struct metablock).

Else , it loops through the memorylist, merging any consecutive blocks.