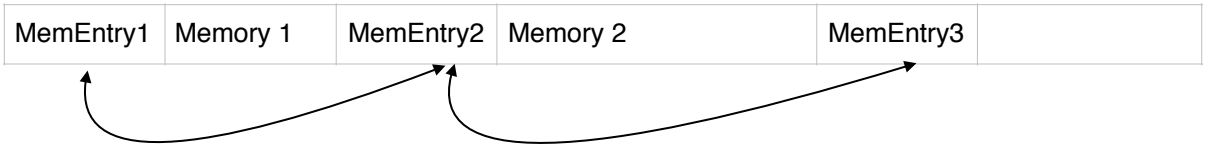
Assignment 3:

Error detecting malloc() and free()

heap.c contains 2 functions named malloc() and free(). They can be used to detect frequent errors made by the users of the real malloc() and free().
By default, the user will be given 5000 bytes to store all the memory. In order to expand the memory the user can change the HEAP_SIZE in heap.c.



The **Data Structure:**

The data structure used to represent one memory entry is doubly linked list where each node is a memory entry with the following variables:

+ MemEntry* next: points to the next memory entry.
+ MemEntry* previous: points to the previous memory entry.
+ int isFree: marks whether the memory entry is freed or not.
+ unsigned size: the size of the memory.

The memory which the user allocates is located consecutively to the memory entry, where every entry does a book-keeping job.

Improved malloc(): Allocate a new chunk of memory and return the address at which the memory can be used by the user without affecting the other data.
The way it does this is to traverse to each memory entry, look for a isFree-ed one which also needs to be large enough to contains the next entry and the size of memory the user needs. If found, malloc() will create a new entry and returns the address the user need to store the data. Otherwise, an error appears stating that there is no free memory. Hence, in the worst case, malloc() goes to the end memory entry and create the new entry there. The efficiency is O(n).

Improved free(): tries to free the memory at the address the user passed as a parameter. This function also traverses through the linked list of entries. At each entry, free() checks whether the address of the allocated data is the address the user passed in. If it is, free() checks the isFreed variable of the entry.

If the entry is freed, an error is reported:

+ Memory is redundantly freed.

Otherwise, free() sets the entry isFreed and tries to merge the current entry with the previous entry or the next entry (if they are already freed).

On the other hand, if the address the user gave lies between the data the user allocated, another error is prompted:

+ Memory freed is not the memory previously returned by malloc().

If no entry contains the address the user gave, free() will go to the end node and report an error:

+ Memory is not pre-allocated by malloc().

Thus, in the worst case, free() will reach the end memory entry. The efficiency of the algorithm is O(n).

In this design, we do not use sbrk() to expand the data segment of the memory as sbrk() is a very “expensive” system call. The user can optionally expand the default size of the “fake” heap by tweaking HEAP_SIZE macro.