Basic Memory Structure:
Before any blocks are initialized, 9 bytes at the start of memory are used in order to keep track of various things in the memory array.

The first byte is used by char I. This character is used to tell if the first block has been initialized.
The next 4 bytes are used by int memory_allocated. This tells us the total amount of allocated memory that is being used currently.
The last 4 bytes is used by int blocks_allocated. This tells us the number of blocks that are currently allocated.

Memory is divided into blocks, with each block contains two parts, a 5 byte header and the actual space that is going to be used as memory for the program. The first byte of the header is char free, a character which tells us whether this block has been freed or not. The next 4 bytes are used by int size, which tells us the size of the allocated memory.

Example:

```
int* x = (int *) malloc(4);
x[0] = 15;
```

after these two statements, memory would look like this:

[1 (15) (1) 0 (4) (15) …]

where numbers is parenthesis symbolize an int, and numbers not in parenthesis symbolize a char.
If we were to free the block now using the statement “free(x);”, then the memory would become:

[1 (14) (1) 1 (4) …]

The function void* malloc follows the following steps:

- Checks to see if there is enough space for allocation
- Tests whether a block has been initialized yet or not
- If not, creates the first block
- If yes, iterates through blocks looking for one large enough
- If it finds a sufficiently large block, allocate the memory and split any excess into another block
- If no block is free and sufficiently large, a new block is created.

Malloc iterates through the blocks using int size located in the header of each block. Also, if the free block after a split will be less than or equal to 5 bytes then instead of splitting the memory is left in the block as excess memory.

The function void free follows the following steps:

- Checks if ptr has been allocated
- If allocated, check if free. If not allocated, report an error (pointer was never allocated)
- If free, report an error (pointer was already freed) if not, free it
- Merge adjacent free blocks

In addition there are two helper functions, void* newblock and void merge. Newblock is used to create a new block at the end of the currently allocated memory in the case of no block being sufficiently

large enough, and merge iterates through every block after a free. If it finds two adjacent blocks that are both free, they are merged into a single larger free block.