Our implementation:

Our memory is stored in a doubley-linked list. We use 8 bytes for metadata. The first 31 bits are reserved for the size of the block. The following bit is used to store the allocation flag. The next 31 bits store an int representing the address of the previous block. The last bit is a flag signifying whether the block is the last one in the list.

To iterate through the list, we use the size of the block to find out the address of the next block.

mymalloc() uses the first free method to assign new blocks. First it checks if there is enough space left to fit the requested memory. Then it traverses the list til it finds an empty block where it fits. Then it either fills the entire empty block, or if there is excess room, inserts itself and splits off a new empty block. It then returns a pointer to the start of the data.

To protect against invalid pointers, free() traverses memory from the first block, and only frees a block if its address matches the given pointer. When freeing a block, it checks the previous and next block: if one or both are unallocated, it will merge with them.

Workload findings: