Lawrence Angrave. CS241 System Programming
Today: Memory Allocation

```
typedef struct _metadata_entry_t {
    void *ptr;
    int size;
    int free;
    struct _metadata_entry_t *next;
} metadata_entry_t;

metadata_entry_t *metadata = NULL;

// TA simple solution. i) Complete the code. ii) Which placement algorithm
does this use? iii) Does this implementation use explicit or implicit linked
list? How would you change this to use a first-fit placement allocation? iv)
Why does this code suffer from false fragmentation?
void *malloc(size_t size) {

    /* See if we have free space of enough size. */
    metadata_entry_t *p = metadata;
    metadata_entry_t *chosen = NULL;

    while (p != NULL) {
        if (p->free && ____________________________) {
            if (chosen == NULL || (chosen && p->size < chosen->size)) {
                chosen = p;
            }
        }
        p = p->next;
    }

    if (chosen) {
        chosen->free = 0;
        return chosen->ptr;
    }

   /* Add our entry to the metadata */
    chosen = sbrk(0);
    sbrk(sizeof(metadata_entry_t));
    chosen->ptr = sbrk(0);
    if (sbrk(size) == (void*)-1) {
        return NULL;
    }
    chosen->size = size;
    chosen->free = 0;

    chosen->next = metadata;
    metadata = chosen;

    return chosen->ptr;
}
```

```
void free(void *ptr) {
// "If a null pointer is passed
//as argument, no action occurs."
    if (!ptr)
        return;

// Free the memory in our metadata.

    metadata_entry_t *p = metadata;
    while (p) {
        if (p->ptr == ptr) {
            p->free = 1;
            return;
        }
        p = p->next;
    }

    return;
}
```

Implementation – Key Ideas

Placement algorithm. Given a linked list of free spaces

Natural Alignment : Platform able to store all standard C primitives at that address. Platform specific but it is typical: malloc(..) % 16 ==0

External Fragmentation: When the available space is not contiguous. Depends on pattern of allocations and frees.
vs
Internal Fragmentation: 'Hidden unused space' inside each allocation
(standard example: round up each allocation request to 2^n => unused space *inside* each block)

1. Implicit linked list: Store size of block and calculate offsets to next block
-> Solving Coalescing Problem "False Fragmentation? Use Knuth73 Boundary Tags so we can coalesce backwards too.
O(N) alloc. O(1) free.

2. Explicit linked list: Store memory addresses of next link
-> Store free blocks pointers inside the unused space of the free block.
-> Free Block list can now be in arbitrary order.

3. Segregated free list: Different lists for different sizes

Advanced implementation ideas: Buddy Allocator, Slab allocator. Deferred coalescing?