CS 341 | Lecture #9
Building an allocator

1. The following allocator will use this linked list structure:

```
typedef struct _metadata_entry_t {
    void *ptr;
    size_t int size;   size of the memory allocated
    int free; //0(in use) or 1(available)
    struct _metadata_entry_t *next;   ← linked list!
} metadata_entry_t;
```

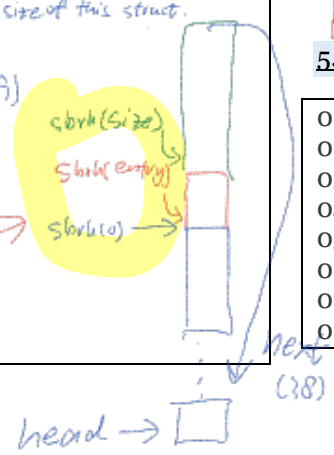
Global variable: ← need an head

```
static metadata_entry_t * head = NULL;
```

2. Complete `malloc()`

```
void *malloc(size_t size) {

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

    while (p != NULL) {
        if (p->free && p->size >= size) {
            if (chosen == NULL || (chosen && p->size < chosen->size)) {
                chosen = p;   using best fit algorithm.
            }
        }
        p = p->next;   where split happens
    }

    if (chosen) {   if chosen get returned, malloc may crash
        chosen->free = 0;
        return chosen->ptr;   ← ptr to actual area.
    }

    /* Add our entry to the metadata */
    chosen = sbrk(0);   ← get top of heap address
    chosen = sbrk(sizeof(metadata_entry_t));   ← increase by size of this struct.
    chosen->ptr = sbrk(0);
    if (sbrk(size) == (void*)-1) {
        return NULL;
    }
    chosen->size = size;
    chosen->free = 0;   entry

    chosen->next = head;
    head = chosen;
    return chosen->ptr;
}
```

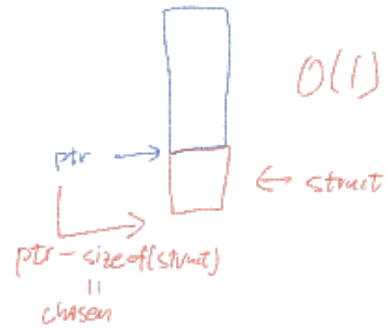
can use chosen



might be wrong
think about this

3. Complete `free()`

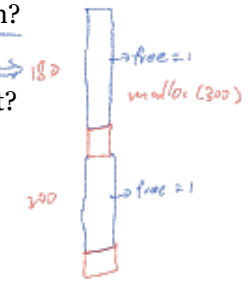
```
void free(void *ptr) {
    if (!ptr) return;


    metadata_entry_t *p = head
    while (p) {
        if (p->ptr == ptr) {
            p → free = 1
            return


        }
        p = p->next;
    }

    return;
}
```

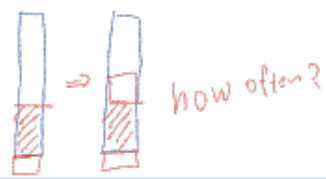


Which placement algorithm does this malloc( )use?
Is calling sbrk 4 tims necessary?
What is the order of growth running time for this implementation of free? O(N)

Interview question

4 i) Why does this implementation suffer from false fragmentation?
ii) When should we split blocks?
iii) Does this implementation use an explicit or implicit linked list?





5. How would you change `malloc()` to use a *first-fit* placement allocation?

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

8. Towards a better allocator

Implementing `realloc` & improving performance of `free()`

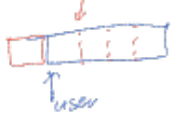
Hint: Can we ensure this structure is immediately before the user's pointer?

```
typedef struct _metadata_entry_t {
  ~~void *ptr~~; since we know the size of the struct.
size_t ~~int~~ size;
  int free; → hide this in another variable !!!
  struct _metadata_entry_t *next;
} metadata_entry_t;
```

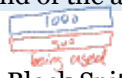
We want an O(1) deallocator!

```
void free(void*user) {
  if(user == NULL) return; // No-op
  ? entry* p = user - sizeof(entry_t)
OR ☆ p = ((entry_t) user) - 1

p -> free = 1
```

convert to entry_t



End of the allocator challenge?



1. Block Spitting & Block Coalescing edge case!!
2. Memory pools
3. Advanced: Slab allocator and Buddy allocator
4. Internal vs External Fragmentation
5. How we use Boundary Tags to implement coalescing?

9. Puzzle:

Complete this code to read in values from stdin into heap memory. Can you beat CS225 code by using C and `realloc` to increase the size of the array? Fix any errors you notice.

```
#define quit(mesg) {puts(mesg); exit(1);}

size_t capacity = 256;
size_t count =  0;
int* data = malloc( capacity );
if( ! data ) quit("Out of memory");

while( !feof(stdin) && !ferror(stdin)) {
  if( count == capacity) {
     capacity *= 2;
  }
  if( fscanf(stdin, "%d", data+count) != 1) break;
  count++;
}
// can now reduce capacity to the number actually read
printf("%d values read",(int) count);
data = realloc(data, count);
```