

uint64_t* because 8 bytes

Function sig for malloc/free

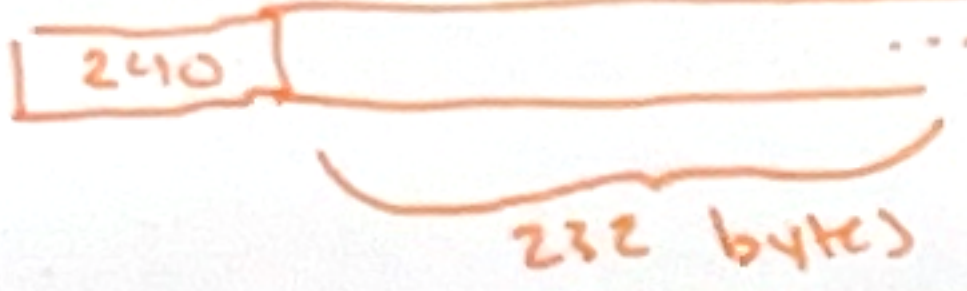
first block on our malloc heap is here

it is a word index

```

1 #include "malloc.h"
2 #include <sys/mman.h>
3 #include <stdio.h>
4 #include <stdint.h>
5
6 uint64_t* HEAP_START = NULL;
7 #define HEAP_SIZE_BYTES 240
8 #define HEAP_WORDS HEAP_SIZE_BYTES/8
9
10 void init_heap() {
11     HEAP_START = mmap(NULL, HEAP_SIZE_BYTES, PROT_READ | PROT_WRITE,
12         MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
13     HEAP_START[0] = HEAP_SIZE_BYTES;
14 }
15 void print_heap() {
16     int i = 0;
17     uint64_t* start = HEAP_START;
18     while(i < HEAP_WORDS) {
19         uint64_t h = start[i];
20         if(h % 2 == 0)
21             printf("%p : %llu [%llu bytes FREE]\n", &start[i], h, h);
22         else
23             printf("%p : %llu [%llu bytes BUSY]\n", &start[i], h, h - 1);
24         i += h / 8;
25     }
26 }
27
28 // Returns a pointer to the header of a block of at least size bytes
29 uint64_t* find_block(uint64_t* start, size_t size);
30
31 /* Splits the block into two, one busy, one free.
32    Note also that the current block size and size are 8-byte aligned */
33 void split_block(uint64_t* block_start, size_t size);
34
35 /* Returns the size of a block for a given malloc request:
36    rounded up to nearest 8, plus 8 for header */
37 size_t block_size_of(size_t user_request);
38
39 void* malloc(size_t size) {
40     if(HEAP_START == NULL) { init_heap(); }
41     size_t block_size = block_size_of(size);
42     uint64_t* block_to_use = find_block(HEAP_START, block_size);
43     if(block_to_use == NULL) {
44         printf("Could not allocate %ld\n", block_size);
45         return NULL;
46     }
47     split_block(block_to_use, block_size);
48     void* addr_for_user = &block_to_use[1];
49     return addr_for_user;
50 }
51
52 void free(void* ptr) {
53     uint64_t* p = ptr;
54     p[-1] = p[-1] & (-1);
55 }

```



free

- align / add

locate a big enough free block

the address the user sees

negative
tilde ~ (bitwise not)

```

1 #include "malloc.h"
2 #include <stdio.h>
3
4 int main() {
5     char* a = malloc(10);
6     char* b = malloc(37);
7     int* c = malloc(7 * sizeof(int));
8     printf("After allocating a, b, c:\n");
9     print_heap();
10    free(b);
11    int* d = malloc(5 * sizeof(int));
12    printf("\nAfter freeing b and allocating d:\n");
13    print_heap();
14 }

```

Round up / align to 16
add 8 for header = 24 bytes
Round to 40, add 8
7*4 = 28 round/align to 32
add 8
20 -> 24 + 8 = 32

```

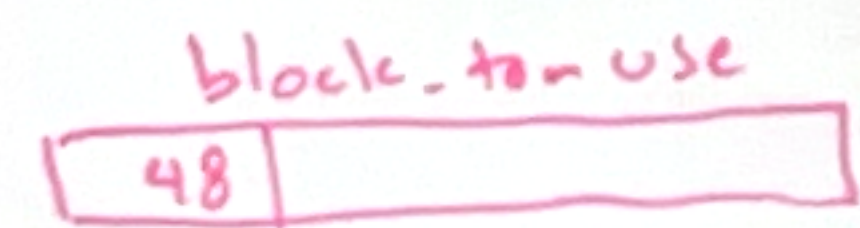
$ gcc -g malloc.c test.c -o test
$ ./test
After allocating a, b, c:
0x1043d4000 : 25 [24 bytes BUSY]
0x1043d4018 : 49 [48 bytes BUSY]
0x1043d4048 : 41 [40 bytes BUSY]
0x1043d4070 : 128 [128 bytes FREE]

After freeing b and allocating d:
0x1043d4000 : 25 [24 bytes BUSY]
0x1043d4018 : 33 [32 bytes BUSY]
0x1043d4038 : 16 [16 bytes FREE]
0x1043d4048 : 41 [40 bytes BUSY]
0x1043d4070 : 128 [128 bytes FREE]

```

do these sizes make sense?

is for d



```
size_t block_size_of(size_t user_request) {
```

```
}
```

```
uint64_t* find_block(uint64_t* start, size_t size) {
```

```
    int i = 0;
```

```
    while(i < HEAP_WORDS) {
```

```
        uint64_t header = start[i];
```

```
        if ((header % 2) == 0 && header >= size) { // First-fit
```

```
            return &start[i];
```

```
        }
```

```
        i += header / 8;
```

```
    }
```

```
    return NULL;
```

```
}
```

```
void split_block(uint64_t* block_start, size_t size) {
```

```
    uint64_t header = block_start[0];
```

```
    if(header == size) {
```

```
        block_start[0] += 1;
```

```
        return;
```

```
    }
```

```
    size_t free_size = header - size;
```

```
    block_start[0] = size + 1;
```

```
    block_start[size / 8] = free_size;
```

```
}
```