uint64_t* because 8 bytes

function sigs for malloc/free

mmap (sbrk)

first block on our malloc heap is here

```c
#include "malloc.h"
#include <sys/mman.h>
#include <stdio.h>
#include <stdint.h>

uint64_t* HEAP_START = NULL;
#define HEAP_SIZE_BYTES 240
#define HEAP_WORDS HEAP_SIZE_BYTES/8

void init_heap() {
  HEAP_START = mmap(NULL, HEAP_SIZE_BYTES, PROT_READ | PROT_WRITE,
      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
  HEAP_START[0] = HEAP_SIZE_BYTES;
}

void print_heap() {
  int i = 0;
  uint64_t* start = HEAP_START;
  while(i < HEAP_WORDS) {
    uint64_t h = start[i];
    if(h % 2 == 0)
      printf("%p : %llu [%llu bytes FREE]\n", &start[i], h, h);
    else
      printf("%p : %llu [%llu bytes BUSY]\n", &start[i], h, h - 1);
    i += h / 8;
  }
}

// Returns a pointer to the header of a free block of at least size bytes
uint64_t* find_block(uint64_t* start, size_t size);

/* Splits the block into two, one busy, one free.
   Note also that the current block size and size are 8-byte aligned */
void split_block(uint64_t* block_start, size_t size);

/* Returns the size of a block for a given malloc request:
   rounded up to nearest 8, plus 8 for header */
size_t block_size_of(size_t user_request);

void* malloc(size_t size) {
  if(HEAP_START == NULL) { init_heap(); }
  size_t block_size = block_size_of(size);
  uint64_t* block_to_use = find_block(HEAP_START, block_size);
  if(block_to_use == NULL) {
    printf("Could not allocate %ld\n", block_size);
    return NULL;
  }
  split_block(block_to_use, block_size);
  void* addr_for_user = &block_to_use[1];
  return addr_for_user;
}

void free(void* ptr) {
  uint64_t* p = ptr;
  p[-1] = p[-1] & (-1);
}
```

2410 ... 232 bytes

a c is a word index

out of memory

negative

tilde ~ (bitwise not)

the address the user sees

block_to_use → 48 | block_size = 32 → 33 | 16

- align/add
- locate a big enough free block

```c
#include "malloc.h"
#include <stdio.h>

int main() {
  char* a = malloc(10);
  char* b = malloc(37);
  int* c = malloc(7 * sizeof(int));
  printf("After allocating a, b, c:\n");
  print_heap();
  free(b);
  int* d = malloc(5 * sizeof(int));
  printf("\nAfter freeing b and allocating d:\n");
  print_heap();
}
```

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]   a
0x1043d4018 : 49 [48 bytes BUSY]   b
0x1043d4048 : 41 [40 bytes BUSY]   c
0x1043d4070 : 128 [128 bytes FREE]

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

do these sizes make sense?

```
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) {  //fast-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;


}
```