```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 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);
}
```

```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();
}
```

```
$ 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 :

0x1043d4018 :

0x1043d4038 :

0x1043d4048 :

0x1043d4070 :
```

```
size_t block_size_of(size_t user_request) {




}

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











}

void split_block(uint64_t* block_start, size_t size) {











}
```