```
#define HEAP_SIZE 4096
#define SLOT_SIZE 8
#define HEAP_SLOTS HEAP_SIZE/SLOT_SIZE

uint64_t* HEAP_START = NULL;

void init_heap() { ... } // set up HEAP_START

size_t block_size(uint64_t s) { return s & (~1); }
int block_busy(uint64_t s) { return s & 1; }
size_t round_size(size_t size) { return (size + 7) & ~7; }



void* allocate_at(uint64_t* start, size_t size) {
  size_t current_size = block_size(start[0]);
  if(current_size > size) {
    uint64_t remaining = current_size - size - SLOT_SIZE;
    int next_block_index = (size / SLOT_SIZE) + 1;
    start[next_block_index] = remaining; // even, free
  }
  start[0] = size | 1; // busy
  return &start[1];
}



void* malloc(size_t requested_size) {
  init_heap();
  int val_index = 0;
  size_t rounded = round_size(requested_size);
  while(val_index < HEAP_SLOTS) {
    uint64_t curr_slot = HEAP_START[val_index];
    int current_size = block_size(curr_slot);
    int current_busy = block_busy(curr_slot);
    if(!current_busy && (current_size >= rounded)) {
      return allocate_at(&HEAP_START[val_index], rounded);
    }
    else {
      val_index += (current_size / SLOT_SIZE) + 1;
      continue;
    }
  }
  return NULL;
}



void free(void* ptr) {







}



int main() {
  int* a = malloc(20);
  int* b = malloc(100);
  int* c = malloc(20);

  free(b);

  int* d = malloc(15);
}
```