CSE 351 Section 9 – Memory Allocation

Exercise 1: Heap Terminology and Behaviors

```
void* ptr1 = malloc(30);
void* ptr2 = malloc(40);
void* ptr3 = malloc(70);
```

a) What pointer is returned if we execute another `malloc` now?

176: the beginning of the free list + 8 bytes for the header

b) Which block(s) could you free that would cause fragmentation in the heap?

The block with the pointer **48**. Neither of the other blocks has an allocated block on both sides.

Note: You could argue that freeing the first block (pointer 8) causes fragmentation if you consider the space before the first allocated block as external fragmentation, but this distinction isn't too important.

c) Which block(s) could you free that would cause coalescing to occur?

Just the block with the pointer **96**. It's the only block bordered by an unallocated block.

d) How many boundary tags do we need to update when calling `free(ptr2)`?

3: the header and footer of the new free block, and the following block's header

e) After calling `free(ptr2)`, which block is at the head of the free list? How many non-null free-list pointers are there?

`ptr2` is the head of the free list (since we add blocks to the head of the free list). There are **2** non-null pointers (`ptr2`'s next, next block's prev).

Exercise 2: Get Block Size

`void* ptr` points to the *payload* of an allocated block. Use the above Lab 5 provided code to **get the size of the allocated block**:

```
size_t size_curr_blk = SIZE( ( (block_info*) UNSCALED_POINTER_SUB(ptr,
                                              WORD_SIZE) )->size_and_tags);
```

Exercise 3: Set prec-used?

`void* ptr` points to the *payload* of an allocated block. Use the above Lab 5 provided code to **set TAG_PRECEDING_USED of the *following* block to True** (can use `size_curr_blk`):

```
block_info* flw_blk = (block_info*) UNSCALED_POINTER_ADD(ptr,
                                    size_curr_blk - WORD_SIZE);

flw_blk->size_and_tags = (flw_blk->size_and_tags) | TAG_PRECEDING_USED;
```

Exercise 4: Copy Tags

Implement the following function. Try using bitwise operators to access the tags in `size_and_tags`.

```
// Copies the tags (TAG_PRECEDING_USED and TAG_USED) from block_to_copy to
// original_block. Leaves the size of original_block unchanged.

void copy_tags(block_info* original_block, block_info* block_to_copy){

  size_t copy_used = (block_to_copy->size_and_tags) & TAG_USED;

  size_t copy_preceding_used = (block_to_copy->size_and_tags) &
                               TAG_PRECEDING_USED;

  original_block->size_and_tags = SIZE(original_block->size_and_tags) |
                                  copy_preceding_used | copy_used;

}
```