Memory allocation service

The goal of this exercise is to create memory allocator in C which is using a big chunk of memory pool which is set at initialization.

You need to implement a **memory allocator** module according to the following API:

```
typedef struct MemoryAllocator MemoryAllocator;

/* memoryPool is a ptr to an already-existing large memory
block */
MemoryAllocator* MemoryAllocator_init(void* memoryPool,
size_t size);

/* Returns a ptr to the memoryPool */
void* MemoryAllocator_release(MemoryAllocator* allocator);

void* MemoryAllocator_allocate(MemoryAllocator* allocator,
size_t size);

/* Merge the next adjacent block is free */
void MemoryAllocator_free(MemoryAllocator* allocator, void*
ptr);

/* Merges all adjacent free blocks, and returns the size of
largest free block */
size_t MemoryAllocator_optimize(MemoryAllocator*
allocator);
```

Algorithm:

1. Each block keeps its **size** and **indication** whether it is free or occupied in a compressed manner (LSB).
2. When allocation is requested, you should find the first free block and do the following:
    - Check if it is big enough. If it is, and it is the exact same size, mark it as allocated and return it. Otherwise, split it to the requested size + the rest, and mark the first block as allocated and the second as free.
    - If it is not big enough, check if the next block is free. If it is, merge them and check the size again.
    - If it is not big enough, look for the next free block.
    - If you reached the end of the memory pool, return NULL.

Restrictions:

Assume that the memory pool is aligned to the platform-specific alignment width (like any allocated memory provided by malloc).

Each allocation must be aligned in the same manner.
The metadata holding the size and the free bit indication should be aligned as well.

Thus, if we are asked to allocate 33 bytes and our platform alignment width is 8 bytes, then we will have a total of 48-bytes block (8 for the metadata, and 40 for the actual allocated data).