Malloc recitation 2 (Fall 2019)

The tar file is located at https://www.cs.cmu.edu/~213/activities/rec11b.tar. Download and extract the tar on a Shark machine, then compile the source:

```
$ wget https://www.cs.cmu.edu/~213/activities/rec11b.tar
$ tar xf rec11b.tar
$ cd rec11b
```

Example 1 (mm.c)

In this example, mm.c is a fake explicit list implementation based on the provided starter code we give for Malloc Lab.

Run the following command:

```
$ gdb --args ./mdriver -c ./traces/syn-array-short.rep -D
(gdb) run
```

Oh no, we get garbled bytes. The best way to determine the cause of them is by using a watchpoint. Use the following commands to do that on the first address that was garbled.

```
(gdb) watch *0x800000010
(gdb) run
```

Keep pressing continue until you reach the mm_malloc.

```
(gdb) c
```

After this point, we see we get garbled bytes at mm.c line 213. This gives us some indication that our problem comes from malloc. To inspect the lines of code at this point, run the following:

```
(gdb) list mm.c:213
```

That looks incorrect. We see that we're overwriting the next and previous pointers of the explicit list which is modifying the data at this point and setting these bytes to 0.

Example 2 (mm-2.c)

Now, let's try mdriver-2.

```
$ gdb --args ./mdriver-2 -c traces/syn-array-short.rep
(gdb) run
```

From running this, you should see an error message saying the payload lies outside of the heap. At some point the header and footer for this block must've been written out of bounds, so we can set a watchpoint at the header corresponding to the payload listed for the error.

```
(gdb) watch *0x8000036c8
(gdb) run
(gdb) continue
(gdb) continue
(gdb) backtrace
```

We see from backtrace that the write comes from the place function which separates the allocated block into the allocated part and free part, adding the free block to the free list. This makes us wonder whether place is incorrect or the value we pass into place is wrong.

From the backtrace, we see that the we provide place with the block parameter equal to 0x8000036c8. This means that when we called place from inside mm_malloc, we gave place a free block which was out of bounds.

This means we can inspect functions which deal with adding free blocks to our list. If we do so, we see that coalesce case 3 sets the block equal to block_next rather than block_prev.