Section 17
Implicit List Heap Allocator
Don’t forget to start recording
## Bird’s Eye View

<table>
<thead>
<tr>
<th>Day</th>
<th>Week 9 Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
<th>Week 10 Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 107A</td>
<td></td>
<td>Section: Implicit List Heap Allocators</td>
<td></td>
<td>Section: Explicit List Heap Allocators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 107</td>
<td>Lecture: Optimization</td>
<td>Lab: Code and Memory Optimization</td>
<td></td>
<td>Lecture: Additional Topics</td>
<td>Lecture: Wrap-up</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 107 assignments</td>
<td>assign5 due, assign6 out</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>assign6 due</td>
<td>Friday</td>
<td></td>
</tr>
</tbody>
</table>

Assignments:
- CS 107A assignments
  - assign5 due, assign6 out
assign6: Heap Allocator
assign6: Heap Allocator

Estimate time spent on assign6: implicit list heap allocator
10 responses

Estimate time spent on assign6: explicit list heap allocator
10 responses
Announcements

- Feedback on weekend OH
- Please start heap allocator early, it’s big
- Heap Allocator Office Hours
  - Friday (?)
  - No OH next week, Slack remains ~asynchronous~
  - Aim to finish implicit heap allocator this week
Agenda

- Implicit List Terminology
- Headers
- Blocks
- Implicit List
Implicit List Terminology
The heap is made up of an implicit list of blocks.
Each block consists of a header and a payload.

<table>
<thead>
<tr>
<th>Req. 1</th>
<th>Free</th>
<th>Req. 2</th>
<th>Free</th>
<th>Req. 3</th>
<th>Free</th>
<th>Req. 4</th>
<th>Free</th>
<th>Req. 5</th>
<th>Free</th>
</tr>
</thead>
</table>

- 0x10  
- 0x11  
- 0x12  
- 0x13  
- 0x14  
- 0x15  
- 0x16  
- 0x17  
- 0x18  
- 0x19  

The highlighted block at 0x14 is in use.
Each header consists of the size and the allocated bit.

<table>
<thead>
<tr>
<th>Req. 1</th>
<th>Req. 2</th>
<th>Req. 3</th>
<th>Req. 4</th>
<th>Req. 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
</tr>
</tbody>
</table>

```
0bxxxxxxxx 0bxxxxxxxx 0bxxxxxxxx 0bxxxxxxxx
0bxxxxxxxx 0bxxxxxxxx 0bxxxxxxxx 0bxxxxxxxx00a
```
Implicit List operations work at 3 levels

Work within the implicit list

Work within a block

Work within a header
Headers

- Required by the handout:
  - 8 bytes, and must be able to hold any size `malloc` takes in
    - So what type should a header be?
Required by the handout:

- 8 bytes, and must be able to hold any size `malloc` takes in
- `typedef size_t header_t;`
Headers

- Required by the handout:
  - 8 bytes, and must be able to hold any size `malloc` takes in
  - `typedef size_t header_t;`
  - Sizes must be a multiple of 8 (use the `ALIGNMENT` constant)
  - An “allocated bit” must be stored within as well
  - Wait, we have to hold 64 bits for the size, and also 1 bit for whether it’s allocated - do we need 65 bits?
Some size_t numbers that are multiples of 8

<table>
<thead>
<tr>
<th></th>
<th>0b00000000</th>
<th>0b00000000</th>
<th>0b00000000</th>
<th>0b00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
<tr>
<td>8</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
<tr>
<td>16</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
<tr>
<td>24</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
</tbody>
</table>
Some size_t numbers that are multiples of 8

<table>
<thead>
<tr>
<th></th>
<th>0b00000000</th>
<th>0b00000000</th>
<th>0b00000000</th>
<th>0b00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
<tr>
<td>8</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
<tr>
<td>16</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
<tr>
<td>24</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
<td>0b00000000</td>
</tr>
</tbody>
</table>
Use the LSB as the allocated bit

- Arbitrarily decide that 0 = FREE, 1 = ALLOC (you can reverse this if you like, as long as you always follow the rule!)

This has an actual value of 25. This value is meaningless.
- We know that when we extract out the size, the LSB should be 0.
- We can extract the actual size by zeroing out the LSB.
- We can extract the allocated bit by reading only the LSB.

\[
\begin{array}{cccccccc}
0b00000000 & 0b00000000 & 0b00000000 & 0b00000000 \\
0b00000000 & 0b00000000 & 0b00000000 & 0b00011001 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0b00000000 & 0b00000000 & 0b00000000 & 0b00000000 \\
0b00000000 & 0b00000000 & 0b00000000 & 0b00011000 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0b00000000 & 0b00000000 & 0b00000000 & 0b00000000 \\
0b00000000 & 0b00000000 & 0b00000000 & 0b00011000 \\
\end{array}
\]

(allocated bit = ALLOC)
Header-level Operations

- **Write:**
  - `bool is_free(header_t *header);` (group)
  - `void set_alloc(header_t *header);` (group)
  - `size_t get_size(header_t *header);` (partner)

- **You may assume:**
  - `typedef size_t header_t;`
  - `the constants FREE and ALLOC are defined (one of them is 1, the other is 0)`
Blocks
Pointers within a Block

- Header pointer (same as block pointer)
- Payload pointer (only used to return to user - you will never write to the payload for the implicit list!)
- Next block pointer
Block-level Operations

- **Write:**
  - `void *header2payload(header_t *header);` *(group)*
  - `header_t *payload2header(void *payload);` *(partner)*
  - `header_t *next_block(header_t *header);` *(partner)*
  - Is it possible to write `prev_block`?

- **You may assume:**
  - The existence of the header-level functions we wrote earlier (don’t examine the values of headers all over again!)
Implicit List
The implicit list must fit the entire heap exactly

- 40, free
- 72, allocated
- 24, free
- 96, allocated
Note: at the very beginning, the heap is just one big giant free block
This is after we’ve done some `malloc`s and `frees`
Overlaps NOT allowed! (header pointer + size must point to next block)

60, free
72, allocated
24, free
96, allocated
The last block must take up exactly the heap remainder (last block pointer + size must be end of heap)

40, free
72, allocated
24, free
96, allocated
Don’t treat these slides as a list of validity characteristics, by the way. Sizes must be a multiple of 8, too, for example.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>free</td>
<td>72</td>
<td>allocated</td>
</tr>
<tr>
<td>24</td>
<td>free</td>
<td>96</td>
<td>allocated</td>
</tr>
</tbody>
</table>
next_block, which we already wrote, helps us iterate through blocks

<table>
<thead>
<tr>
<th></th>
<th>40, free</th>
<th>72, allocated</th>
<th>24, free</th>
<th>96, allocated</th>
</tr>
</thead>
</table>

Implicit list-level Operations

- Write:
  - `size_t count_blocks(header_t *start);` (group)
  - `header_t *find_block_of_64(header_t *start);` (group)

- You may assume:
  - The existence of the block-level functions we wrote earlier (don’t parse pointers all over again!)