Allocating and freeing blocks with malloc and free:

1. word = 4 bytes
2. block = 2 words
3. double word aligned
4. free blocks.

1. \( p_1 = \text{malloc}(2 \times \text{sizeof(int)}) \);
2. \( p_2 = \text{malloc}(3 \times \text{sizeof(int)}) \);

\( p_1 \)

\( p_2 \)

wasted.
3. \texttt{p3 = malloc(4 * sizeof(int))};

4. \texttt{free(p2)};

5. \texttt{p4 = malloc(2 * sizeof(int))};
Allocator Requirements

1. Handling arbitrary request sequences.
   - Requests should not be re-ordered.
   - "..." buffered.

2. Making immediate responses to requests.

3. Using only the heap
   Heap management data structures can use only the heap memory.

4. Aligning blocks (alignment requirement).
   Blocks should be able to hold any type of data object.
   On most systems, blocks are aligned on a 8-byte boundary.
   Double word

5. Not modifying allocated blocks.
   * Cannot modify or move allocated blocks.
   ⇒ Compaction of allocated blocks - NOT ALLOWED.
Allocator Goals

1. Maximizing throughput
   Throughput - no. of requests the allocator completes in a unit time.
   
   \[ \text{e.g., if 500 allocate requests and 500 free requests in one sec.} \]
   \[ \Rightarrow \text{throughput} = 1000 \text{ operations per sec.} \]

2. Maximizing memory utilization
   Virtual memory is not an infinite resource.

\[
\begin{align*}
\text{malloc} \left( 2 \times \text{sizeof(int)} \right) \\
\text{malloc} \left( 4 \times \text{sizeof(int)} \right)
\end{align*}
\]

\[
\begin{align*}
\text{memory utilization} &= \frac{\text{total amount of memory allocated to the application}}{\text{total heap size}}
\end{align*}
\]
Fragmentation

* primary cause for poor heap utilization.

1. **Internal fragmentation** - allocated block is larger than the payload.

\[ p_1 = \text{malloc}(1 \times \text{sizeof(int)}) ; \]
\[ p_2 = \text{malloc}(3 \times \text{sizeof(int)}) ; \]

2. **External fragmentation.**

\[ p = \text{malloc}(4 \times \text{sizeof(int)}) ; \]

will fail!
Implementation Issues

1. Free block organization
2. Placement
3. Splitting

Simple Heap block

malloc returns a pointer to the beginning of the payload.

```
<table>
<thead>
<tr>
<th>Block size</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Header</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Padding (optional)</td>
</tr>
</tbody>
</table>
```

Block size = header + payload + padding.

min block size = 4 + 4 = 8 bytes.
Question: How do we know how much memory to free just given a pointer?

Answer: Keep the length of a block in a "header" preceding the block. One extra word needed for every allocated block.

\[ p = \text{malloc (4)} \]
Free block Organization

1. Implicit free list
2. Explicit free list

Implicit Free List

Idea: Each block contains a header with some extra information.

No explicit structure tracking location of free or allocated blocks.

<table>
<thead>
<tr>
<th>8/0</th>
<th>16/1</th>
<th>16/0</th>
<th>16/1</th>
<th>0/1</th>
</tr>
</thead>
</table>

⇒ double word aligned.

minimum block size = 4 bytes + 4 bytes (header) (alignment requirement)

⇒ malloc(1) would need a 2-work block (8 bytes).
\[
\text{malloc (20)} \quad \xrightarrow{\quad} 4 + 20 + 0 = 24 \text{ bytes}
\]

\[
\begin{align*}
\quad & \quad \text{header} \downarrow \quad \text{payload} \downarrow \quad \text{padding} \\
\therefore \quad & \quad \text{block size} = 0x18 \quad \left(24_{10}\right) \\
\text{allocated bit} = 0x1
\end{align*}
\]

\[
\text{Header} = \quad \text{block size} \quad \left| \quad \text{allocated bit}ight.
\]

\[
\begin{align*}
\text{Header} &= 0x18 \quad | \quad 0x1 \\
&= \quad 0001 \quad 1000 \quad | \quad 0001 \\
&= \quad 0001 \quad 1001 \\
&= \quad 0x19
\end{align*}
\]

\[
\text{Header (4 bytes)} = 0x00 \quad 00 \quad 00 \quad 00 \quad 19
\]
Header of a free block with a block size of 40 bytes (0x28).

$0x\ 00\ 00\ 00\ 28 |\ 0x\ 0 = 0x\ 00\ 00\ 00\ 28$.

Advantage: Simplicity!

Disadvantage: Cost of placing allocated blocks = linear in the total no. of allocated and free blocks in the heap.

2. Placing Allocated Blocks

× Placement policy
1. First fit
2. Next fit
3. Best fit

Best fit - Better memory utilization when compared to first fit or next fit.

- Disadvantage: Exhaustive search of the heap.
3. Splitting Free Blocks

malloc (4)

This block has been split into 2 blocks.

allocated  free
4. Coalescing Free Blocks

\[ \text{malloc (3 * size of (int))} \]

\[ \Rightarrow \text{allocate 3 words.} \]

\[ \Rightarrow \text{CANNOT be done in the above heap.} \]

\[ \Rightarrow \text{False Fragmentation!} \]

\[ \Rightarrow \text{We need to coalesce free blocks.} \]

Now space available for 3 words!
2 types of coalescing

1. immediate coalescing (used in P5)
2. deferred coalescing

Coalescing with Boundary Tags

Heap block with a boundary tag

```
Block size  0 0 a
```

```
Payload
```

```
Padding
```

```
Block size  0 0 a
```

\[ a = 1 \implies \text{allocated} \]
\[ a = 0 \implies \text{free} \]

How to coalesce the previous block?

Coalescing with next block is straightforward.

```
8/0 16/0
```

\[ 8 + 16 = 24 \]

```
24/0
```
How to do this?
Add a footer (similar to the header) in the heap block.

4 cases: prev & next allocated

1. \[ \begin{array}{c|c}
    m_1 & a \\
    m_2 & a \\
    n & a \\
    m_{-1} & a \\
\end{array} \]
   \[ \rightarrow \]
   \[ \begin{array}{c|c}
    m_1 & a \\
    m_2 & a \\
    n & f \\
    m_{-1} & a \\
\end{array} \]

2. \[ \begin{array}{c|c}
    m_1 & a \\
    m_2 & f \\
    n & a \\
\end{array} \]
   \[ \rightarrow \]
   \[ \begin{array}{c|c}
    m_1 & a \\
    m_2 & f \\
    n+m_2 & f \\
\end{array} \]

3. prev free, next allocated

4. prev & next is free

See fig 9.40 in CS:APP