Types of cache questions:
1) TIO Breakdown
2) For fixed cache parameters, analyze the performance of the given code/sequence
3) For fixed cache parameters, find best/worst case scenarios
4) For given code/sequence, how does changing your cache parameters affect performance?
5) Average Memory Access Time (AMAT)

What are the important cache parameters?
- Must figure these out from problem description
- Address size, cache size, block size, associativity, replacement policy
- Solve for TIO breakdown, # of sets, management bits
- What starts in the cache?

Code data structures affect addresses of memory accesses
- Array elements are stored contiguously in memory
  - Ideal for spatial locality – if used properly
  - Different arrays not necessarily next to each other in memory
- Structs and linked list nodes are stored separately in memory
  - Addresses of nodes/structs may be very different
  - Method of linking between nodes and ordering of nodes are important
- Remember to account for data size (in bytes)!
  - `char` is 1, `int/float` is 4, `long/double` is 8, `struct` is ?
- Pay attention to access pattern of code
  - `Arrays`: does code touch all elements (e.g. increment all elements, compute array sum) or just some elements (e.g. stride-by-N)?
  - `Linked Lists`: generally starting from “head” node, how many struct elements are touched?

Cache access patterns
- How many hits within a single cache block once it is loaded into the cache?
- Will that cache block still be in cache when you revisit its elements?
- Are there special/edge cases to consider?
  - Usually edge of block boundary or edge of cache size boundary
Example Cache Question: Assume our processor has two levels of data caches with the parameters shown in the table below. Also assume:
- 1 GiB address space
- 100 clock cycles to access data in memory

a) Fill in the rest of the table below:

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>32 KiB</td>
<td>512 KiB</td>
</tr>
<tr>
<td>Block Size</td>
<td>8 B</td>
<td>32 B</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
<td>Direct-mapped</td>
</tr>
<tr>
<td>Hit Time</td>
<td>1 cycle</td>
<td>33 cycles</td>
</tr>
<tr>
<td>Miss Rate</td>
<td>10%</td>
<td>2%</td>
</tr>
<tr>
<td>Write Policy</td>
<td>Write-through</td>
<td>Write-through</td>
</tr>
<tr>
<td>Replacement Policy</td>
<td>LRU</td>
<td>n/a</td>
</tr>
</tbody>
</table>

For the rest of this problem, we decide to use ONLY the L1$ and run the following code. Assume:
- The cache starts cold
- The array char A[] is block-aligned and SIZE = 2^25 (32 MiB array)
- Variables i, j, sum, and prod are stored in registers

```c
char *A = (char *) malloc (SIZE * sizeof(char));

for (i=0; i<(SIZE/STRETCH); i++) { // # of STRETCHes
    // go up to STRETCH
    for (j=0; j<STRETCH; j++) sum += A[i*STRETCH + j];
    // down from STRETCH
    for (j=STRETCH-1; j>=0; j--) prod *= A[i*STRETCH + j];
}
```

a) As we double our STRETCH from 1 to 2 to 4 (... etc), we notice the number of cache misses doesn’t change! What is the largest value of STRETCH before cache misses changes?

b) If we double the STRETCH from part (b), what is the ratio of cache hits to misses?