struct Student {
    int id;
    char* name;
    char age;
};

a) Fill in which bytes are used by which variables and label the rest as internal or external fragmentation. The first variable “id” is given.

b) What is the size of struct Student? 24 bytes

c) Give a reordering of the fields in struct Student such that there is no internal fragmentation

struct Student {
    char* name;
    int id;
    char age;
};

d) How much external fragmentation does this new struct Student have? 3 bytes

e) What is the size of this new struct Student? 16 bytes (smaller than before)
We have a two-dimensional matrix of integer data of size \( M \) rows and \( N \) columns. We are considering 3 different representation schemes:

1) 2-dimensional array `int array2D[][],` // \( M \times N \) array of ints
2) 2-level array `int* array2L[], and` // \( M \) array of int arrays
3) array of linked lists `struct node* arrayLL[].` // \( M \) array of linked lists (struct node)

Consider the case where \( M = 3 \) and \( N = 4 \). The declarations are given below:

<table>
<thead>
<tr>
<th>2-dimensional array:</th>
<th>2-level array:</th>
<th>Array of linked lists:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int array2D[3][4];</code></td>
<td><code>int r0[4], r1[4], r2[4]; int* array2L[] = (r0, r1, r2);</code></td>
<td><code>struct node { int col, num; struct node* next; };</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>struct node* arrayLL[3];</code></td>
</tr>
</tbody>
</table>

For example, the diagrams below correspond to the matrix:

```

```

For array2L and arrayLL:

a) Fill in the following comparison chart:

<table>
<thead>
<tr>
<th></th>
<th>2-dim array</th>
<th>2-level array</th>
<th>Array of LLs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Memory Used</td>
<td>( M \times N ) * sizeof(int) = 48 B</td>
<td>( M \times N ) * sizeof(int) + ( M ) * sizeof(int*) = 72 B</td>
<td>( M ) * sizeof(struct node*) + ( M \times N ) * sizeof(struct node) = 216 B</td>
</tr>
<tr>
<td>Largest guaranteed continuous chunk of memory</td>
<td>The whole array (48 B)</td>
<td>The array of pointers (24 B) &gt; row array (16 B)</td>
<td>The array of pointers (24 B) &gt; struct (16 B)</td>
</tr>
<tr>
<td>Smallest guaranteed continuous chunk of memory</td>
<td>The whole array (48 B)</td>
<td>Each row array (16 B)</td>
<td>Each struct node (16 B)</td>
</tr>
<tr>
<td>Number of memory accesses to get int in the BEST case</td>
<td>1</td>
<td>2</td>
<td>First node in LL: 2</td>
</tr>
<tr>
<td>Number of memory accesses to get int in the WORST case</td>
<td>1</td>
<td>2</td>
<td>Last node in LL: 5 (we have to read <code>next</code>)</td>
</tr>
</tbody>
</table>

b) Sam Student claims that since our arrays are relatively small (\( N < 256 \)), we can save space by storing the `col` field as a `char` in `struct node`. Is this correct? If so, how much space do we save? If not, is this an example of internal or external fragmentation?

No. Alignment requirement of \( K = 4 \) for `int num` leaves 3 bytes of internal fragmentation between `col` and `num`. 
c) Provide a scenario where a 2-dimensional array would be more useful and another where a 2-level array would be more useful.

- **2D Array** - Creating a table or a matrix where all rows are the same size. This way memory accesses are reduced and less memory is required.

- **2-Level Array** - When creating a list where different index sizes differ or sub-arrays are subject to replacement. In other words, when the array is more flexible to changes.

d) Sam wants to create a 2-D matrix of the countries of the world that can be accessed alphabetically. Which implementation should Sam choose to represent this information? Describe what this implementation would look like.

Sam should use a 2-level array since the amount of countries starting with a given letter will vary (i.e. there are more countries that start with A than Q). He could make an array of pointers from 0 to 25 which would point to custom-sized arrays of country names starting with each corresponding letter of the alphabet.