- **Array Allocation** \( T \ arr[N] \) Creates an array of \( N \) elements of type \( T \)
  - Elements are lined up in a contiguous section of memory.
  - The array acts as a pointer (you can dereference it, do pointer math, etc) but does not live in memory as a pointer
    - \( arr[0] \) is equivalent to \(*arr\)
    - \&\( arr[i] \) = \( arr+i*\text{sizeof(\text{element})} \)
    - leal was built for array accesses!
  - No bounds checking on arrays (you can get a negative or way-too-large element)
    - Sometimes this creates a segfault, sometimes it just returns a junk value in memory
- **Nested Array Allocation** \( T \ arr[N][M] \) Creates a two-dimensional array of \( N*M \) elements
  - If \( N \) is the number of rows, we have row-major memory layout:
    - Elements in a row \( arr[i][0] \ldots arr[i][M-1] \) are guaranteed to be next to each other in mem
    - Rows are sequential in memory
    - Total size is \( N*M*\text{sizeof(\text{element})} \)
    - Get to an element \( arr[i][j] \) with \( arr+i*N*\text{sizeof(\text{element})}+j*\text{sizeof(\text{element})} \)
  - **Row Vectors** allow you to access a single row of a nested array as if it were its own array
    - Declare \( arr[N][M] \) —> \( \text{foo} = arr[1] \) is equivalent to having \( \text{foo}[M] \) (indexed the same way)