- **Multi-level arrays: array of pointers to other arrays**
  
  ```
  T arr1[M];
  T arr2[M];
  T *mlarray[2] = {arr1, arr2}
  ```

  - `arr1` and `arr2` may not (probably not) contiguous in memory.
  - guaranteed to have all elements of `arr1` together, and all elements of `arr2`
  - `mlarray[0]` is a pointer to `arr1`

- **Structures**

  ```
  struct s1 { int i; int a[3]; int *p; };
  ```

  - Contiguously allocated section of memory
  - Fields are referred to by name (May have different types/sizes)
  - access `s1.i` or `s1.a[1]`
    - `&s1.i + sizeof(i) = &s1.a[0]`

  - Have to add padding to satisfy alignment:
    - All struct fields must start on a multiple of their size
      - leads to padding between fields (internal fragmentation)
    - All structs must have a total size that is a multiple of their largest field
      - leads to padding at the end (external fragmentation)
    - Can minimize padding by ordering fields by their size
  
- **Arrays of structs**

  - line structs up in-place in memory (but still include the padding!)

- **Unions**

  ```
  union U { char c, int i[2], double v }
  ```

  - Are like multiple choice: you can have `c` OR `i` OR `v`, but not more than one
  - Size is the size of the biggest element (wasted space if you use store data in smaller form)
  - Can store data in one format, and get it out in another format
    - Watch out for endianness!