2-D array

- Logically 2-D array represents m x n matrix i.e. m rows and n columns.

```
int arr[3][4] = { {1, 2, 3, 4}, {10, 20, 30, 40}, {11, 22, 33, 44} };
```

- Array declaration:

```
int arr[3][4] = { {1, 2, 3, 4}, {10, 20, 30, 40}, {11, 22, 33, 44} };
int arr[3][4] = { {1, 2 }, {10}, {11, 22, 33 } };
int arr[3][4] = { 1, 2, 10, 11, 22, 33 };
int arr[ ][4] = { 1, 2, 10, 11, 22, 33 };
```

- 2-D array is collection of 1-D arrays in contiguous memory locations.
  - Each element is 1-D array.
- Pointer to array is pointer to 0th element of the array.
  - Scale factor of the pointer = number of columns * sizeof(data-type).

```
int arr[3][4] = { {1, 2, 3, 4}, {10, 20, 30, 40}, {11, 22, 33, 44} };
int (*ptr)[4] = arr;
```

- 2-D array is passed to function by address.
- It can be collected in formal argument using array notation or pointer notation.
- While using array notation, giving number of rows is optional. Even though mentioned, will be ignored by compiler.

Dynamic memory allocation

- Dynamic memory allocation allow allocation of memory at runtime as per requirement.
- This memory is allocated at runtime on Heap section of process.
- Library functions used for Dynamic memory allocation are
  - malloc() - allocated memory contains garbage values.
  - calloc() - allocated memory contains zero values.
  - realloc() - allocated memory block can be resized (grow or shrink).
- All these function returns base address of allocated block as void*.
- If function fails, it returns NULL pointer.

Memory leakage

- If memory is allocated dynamically, but not released is said to be "memory leakage".
- Such memory is not used by OS or any other application as well, so it is wasted.
- In modern OS, leaked memory gets auto released when program is terminated.
- However for long running programs (like web-servers) this memory is not freed.

- More memory leakage reduce available memory size in the system, and thus slow down whole system.

```
int main() {
    int *p = (int*) malloc(20);
    int a = 10;
    // …
    p = &a; // here addr of allocated block is lost, so this memory can never be 
freed.
    // this is memory leakage
    // …
    return 0;
}
```

- In Linux, valgrind tool can be used to detect memory leakage.

Dangling pointer

- Pointer keeping address of memory that is not valid for the application, is said to be "dangling pointer".
- Any read/write operation on this may abort the application. In Linux it is referred as "Segmentation Fault".
- Examples of dangling pointers
    - After releasing dynamically allocated memory, pointer still keeping the old address.
    - Uninitialized (local) pointer
    - Pointer holding address of local variable returned from the function.
- It is advised to assign NULL to the pointer instead of keeping it dangling.

```
int main() {
    int *p = (int*) malloc(20);
    // …
    free(p); // now p become dangling
    // …
    return 0;
}
```