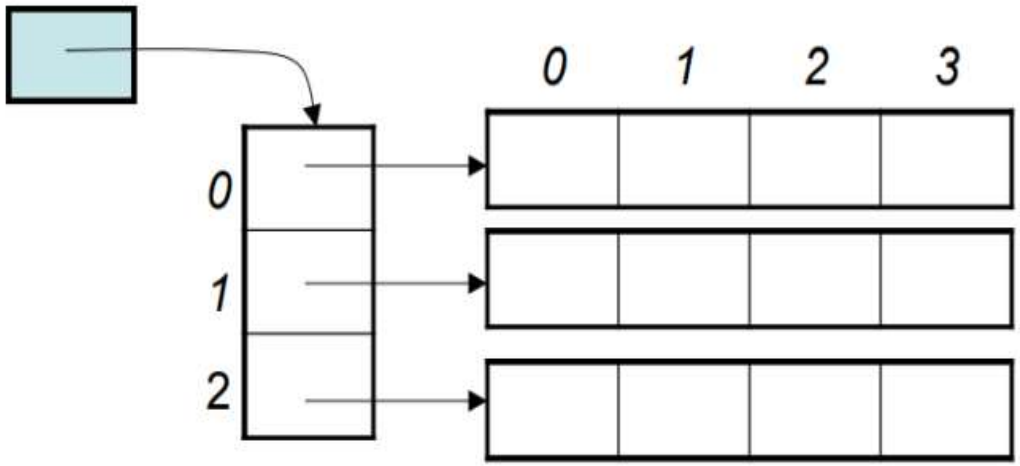
C programming

Trainer : Nisha Dingare

Email : nisha.dingare@sunbeaminfo.com

2-D arrays

- Example: int a[3][5];
  - Logically it may be viewed as a two-dimensional collection of data, three rows and five columns, each location is of type int.



	0	1	2	3	4
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

- **A 2D array is a 1D array of (references to) 1D arrays.**

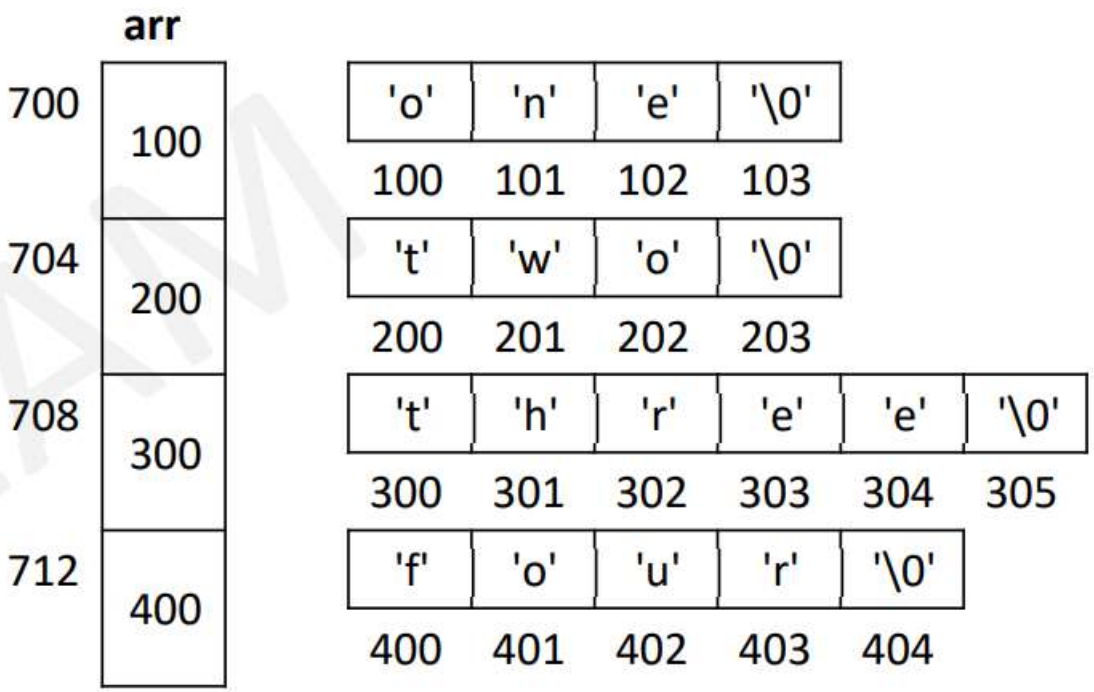
2-D Arrays :

- 2-D array is collection of 1-D arrays in contiguous memory locations.
  - Each element is 1-D array.
- int arr[3][4] = { {1, 2, 3, 4}, {10, 20, 30, 40}, {11, 22, 33, 44} };

	0				1				2			
	0	1	2	3	0	1	2	3	0	1	2	4
arr	**1**	**2**	**3**	**4**	**10**	**20**	**30**	**40**	**11**	**22**	**33**	**44**
	400	404	408	412	416	420	424	428	436	440	444	448
	400				416				436			

Array of pointers :

```
char *arr[] = { "one", "two", "three", "four" };
for(i = 0; i < 4; i++)
        puts(arr[i]);
```



Passing 2-D array to functions :

- 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.

Command Line Arguments :

- The C language provides a method to pass parameters to the main() function.
- This is typically accomplished by specifying arguments on the operating system command line.
- The prototype for main() looks like:
  - int main(int argc, char *argv[]) { … }
  - The first parameter is the number of items on the command line (int argc).
  - argc always retains the count of arguments passed to main
  - Each argument on the command line is separated by one or more spaces, and the operating system places each argument directly into its own null-terminated string.
    - Note: The name of the program is counted and is the first value.
    - Note: Values are defined by lists of characters separated by whitespace.
  - The second parameter passed to main() is an array of pointers to the character strings containing each argument (char *argv[]).
  - Argv catches actual arguments passed at command prompt to main function
    - Note: The array has a length defined by the number_of_args parameter.

- If we add **char **env** an argument to main it will display list of environment variables.
- Environment variables are used for information about your home directory, terminal type, and so on; you can define additional variables for other purposes. The set of all environment variables that have values is collectively known as the *environment.*

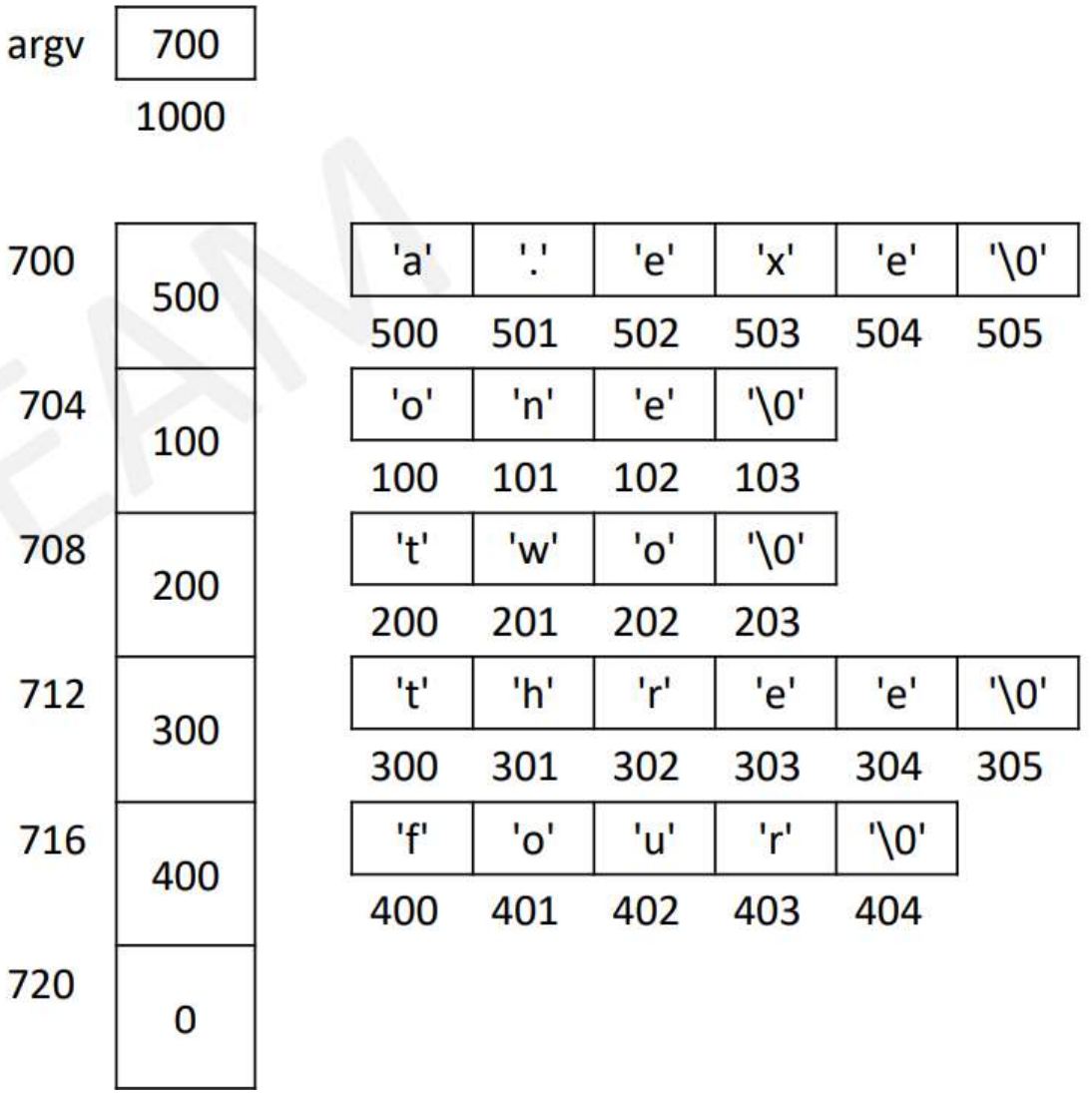
Standard main() prototypes :

- int main();
- int main(int argc, char *argv[]);
- int main(int argc,char*argv[],char*env[]);
- argc represents number of arguments passed to program when it is executed from command line.
- argv represents argument vector or argument values.
- envp represents system information.

Command Line Arguments :

- Command line arguments are information passed to the program while executing it on command line.
- cmd> a.exe one two three four

```
int main(int argc, char *argv[]) {
        int i;
        for(i=0; i < argc; i++)
                puts(argv[i]);
        return 0;
}
```



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.
- The memory block allocated by malloc() has a garbage value.
- The memory block allocated by calloc() is initialized by zero.

Dynamic Memory Allocation :

- Dynamic memory can be requested using malloc,calloc,realloc function
- Such requested memory will be in control of user programmer.
- On demand programmer request memory utilse same and once job is finished programmer has to release such dynamic memory.
- We can shrink or grow dynamic memory at runtime.
- malloc,calloc,realloc function provides memory from heap section.
- If request is successful they will return base address of memory else return NULL.
- void* malloc(int size);
- void* calloc(int count,int ele size);
- void* realloc(void *mem block,int size);
- The address returned by these functions should be type-casted to the required pointer type.

Difference between malloc() and calloc() :

S.No.	malloc()	calloc()
1.	malloc() function creates a single block of memory of a specific size.	calloc() function assigns multiple blocks of memory to a single variable.
2.	The number of arguments in malloc() is 1.	The number of arguments in calloc() is 2.
3.	malloc() is faster.	calloc() is slower.
4.	malloc() has high time efficiency.	calloc() has low time efficiency.
5.	The memory block allocated by malloc() has a garbage value.	The memory block allocated by calloc() is initialized by zero.
6.	malloc() indicates memory allocation.	calloc() indicates contiguous allocation.

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

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;
}
```

Thank you!!