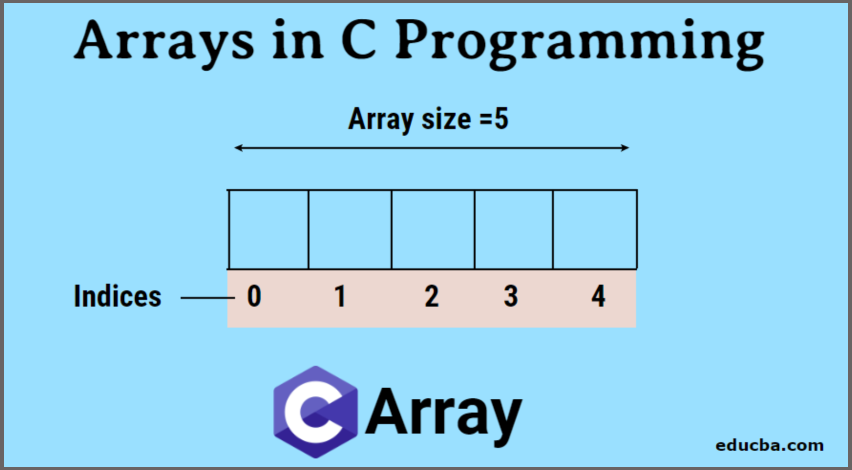
Arrays in C

Lecture 08 – ICT1132



Piyumi Wijerathna
Department of ICT
Faculty of Technology

Overview

- ✓ Introduction to Arrays
- ✓ Defining Arrays
- ✓ Using Arrays
- ✓ Multi-dimensional Arrays
- ✓ Arrays with Functions
- ✓ String Handling

A data type is called *simple* if *variables* of that type can store only one value at a time.

`int count` Enough memory for an `int`

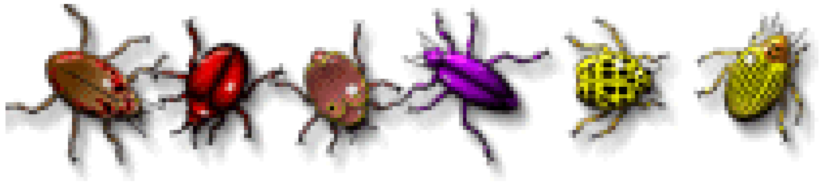
$\boxed{\texttt{12345}}$

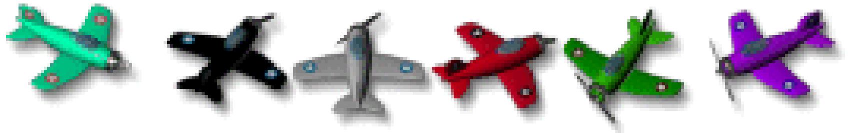
`float price` Enough memory for a `float`

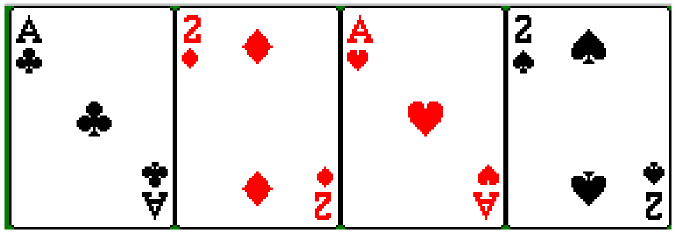
$\boxed{\texttt{56.981}}$

`char letter` Enough memory for a `char`

$\boxed{\texttt{A}}$


An array of bugs


An array of airplanes

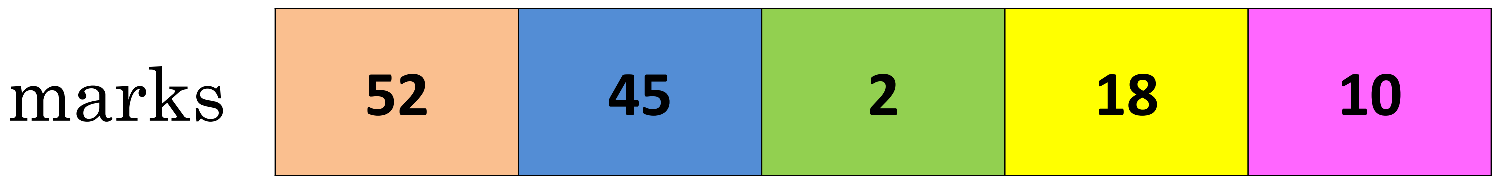

An array of cards

Salvadorian

An array of characters

Introduction

- Arrays: a type of data structure that stores a **fixed number** of data elements of the same **same type.**



- Array stores all the data items sequentially in continuous memory locations.
- All these elements are sharing a same name, and it is the name of the array.
- Size of the array can not be changed later.

Why we need Arrays?

- Imagine keeping track of 5 test scores, or 100, or 1000 in memory
  - How would you name all the variables?
  - How would you process each of the variables?

We can use Arrays to..

- Hold collections of input data for processing.
- Hold other computed values which are needed for processing.

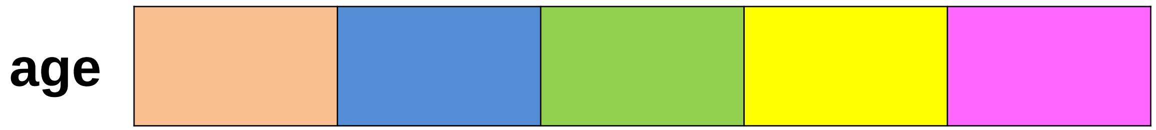
Array Declaration

When declaring arrays, specify

- Name of the array
- Type of array
- Number of elements

data_type array_name[array_size];

```
int age[ 5 ];  // array of 5 age values
```

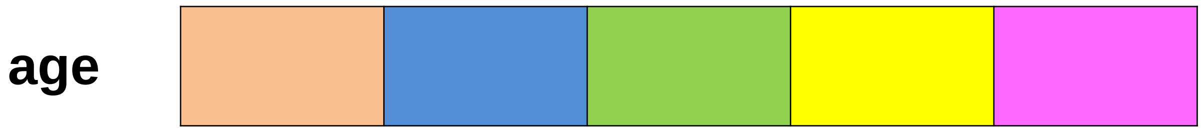


Declaring multiple arrays of same type

- Use comma separated list, like regular variables

```
int a[ 100 ], b[ 27 ];
```

- The **size and type** of arrays **cannot be changed** after its declaration.
- Arrays occupy space in memory.
- You specify the type of the elements and the number of elements, so that the computer may reserve the amount of memory required for the array.

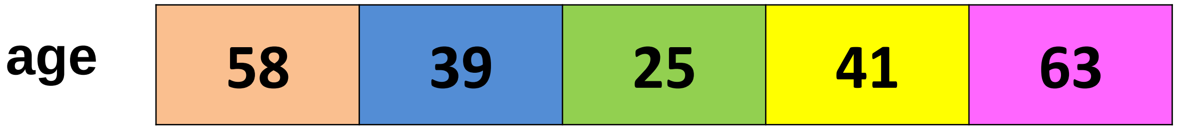


- The previous declaration reserves 16bit * 5 memory space for the array ‘age’.

Array Initialization

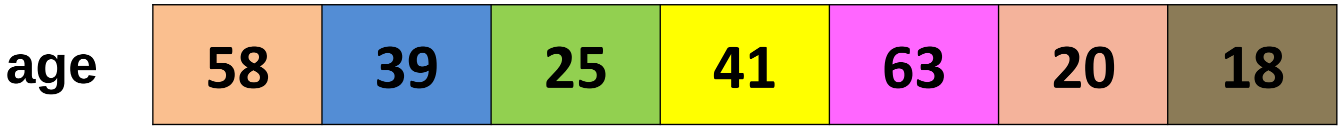
- Array elements can be initialized in two ways during the declaration.

1. int age[5] = {58, 39, 25, 41, 63};



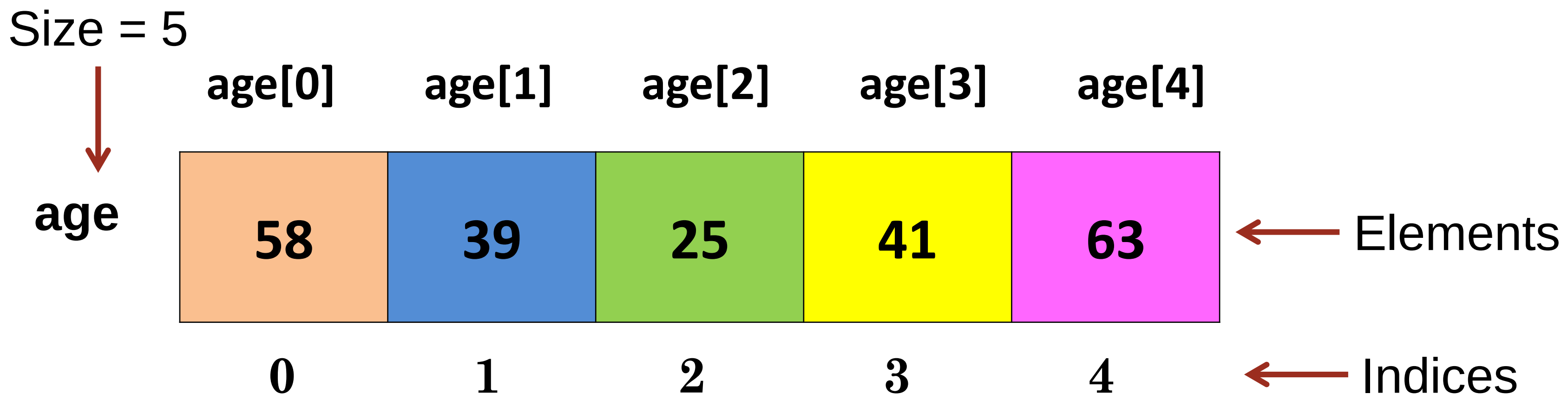
- Extra values result in a syntax error.

2. int age[ ] = {58, 39, 25, 41, 63, 20, 18};



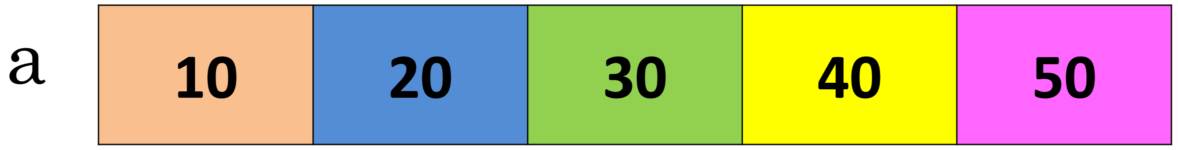
Using Arrays

- Index(subscript) represents the position of an element within the array.
- A specific element/location in an array can be accessed by its name followed by the index within square brackets.



- The 1st element is at the 0th index
- The ending index of any array is (Size – 1)

Using Arrays



- Print the first element of the array

```
printf("%d", a[0] );
```

- Print the sum of first three elements

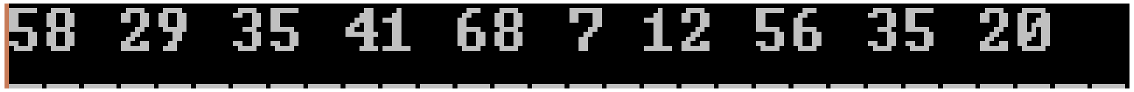
```
printf("%d", a[0]+a[1]+a[2] );
```

- Update the value of 3rd element by adding 5

```
a[2] += 5;
```

Example: Initializing during declaration & print the values

```
# include<stdio.h>
void  main()
{
        // declare & initialize the array
        int age[10] = {58,29,35,41,68,7,12,56,35,20};
        int  x; // counter variable

        //printing the array elements
        for( x = 0; x < 10; x++){
                printf("%d", age[ x ]);
        }
}
```



```
58 29 35 41 68 7 12 56 35 20
```

Declaring Array Size

data_type array_name[size];

The ***size*** must be a constant integer, or an expression which evaluates as a constant integer.

Examples:

a)
```
int myList[5];
```

b)
```
#define MaxSize 10
float numList[MaxSize];
```

c)
```
#define MaxSize 10
float numList[MaxSize+2];
```

Example: Initializing after declaration & size defined by a constant

Score[0]	0.5
Score[1]	1.5
Score[2]	2.5
Score[3]	3.5
Score[4]	4.5
Score[5]	5.5

```
# include <stdio.h>
# define SIZE 6
void  main(){
        float score[ SIZE ];
        int j;
        //storing array elements
        for( j = 0; j < SIZE; j++ ){
                score[ j ] = j + 0.5;
        }
        //printing the array elements
        for( j = 0; j < SIZE; j++ ){
         printf("%.1f\n",score[ j ]);
        }
}
```

Remember!!

- Declared an array of 10 elements.

  int test[10];

- You can use the array members from

  test[0] to test[9].

- If you try to access array elements outside of its bound, ex: test[15],

the compiler may not show any error. Instead it may cause an unexpected output (undefined behavior).

Remember!!

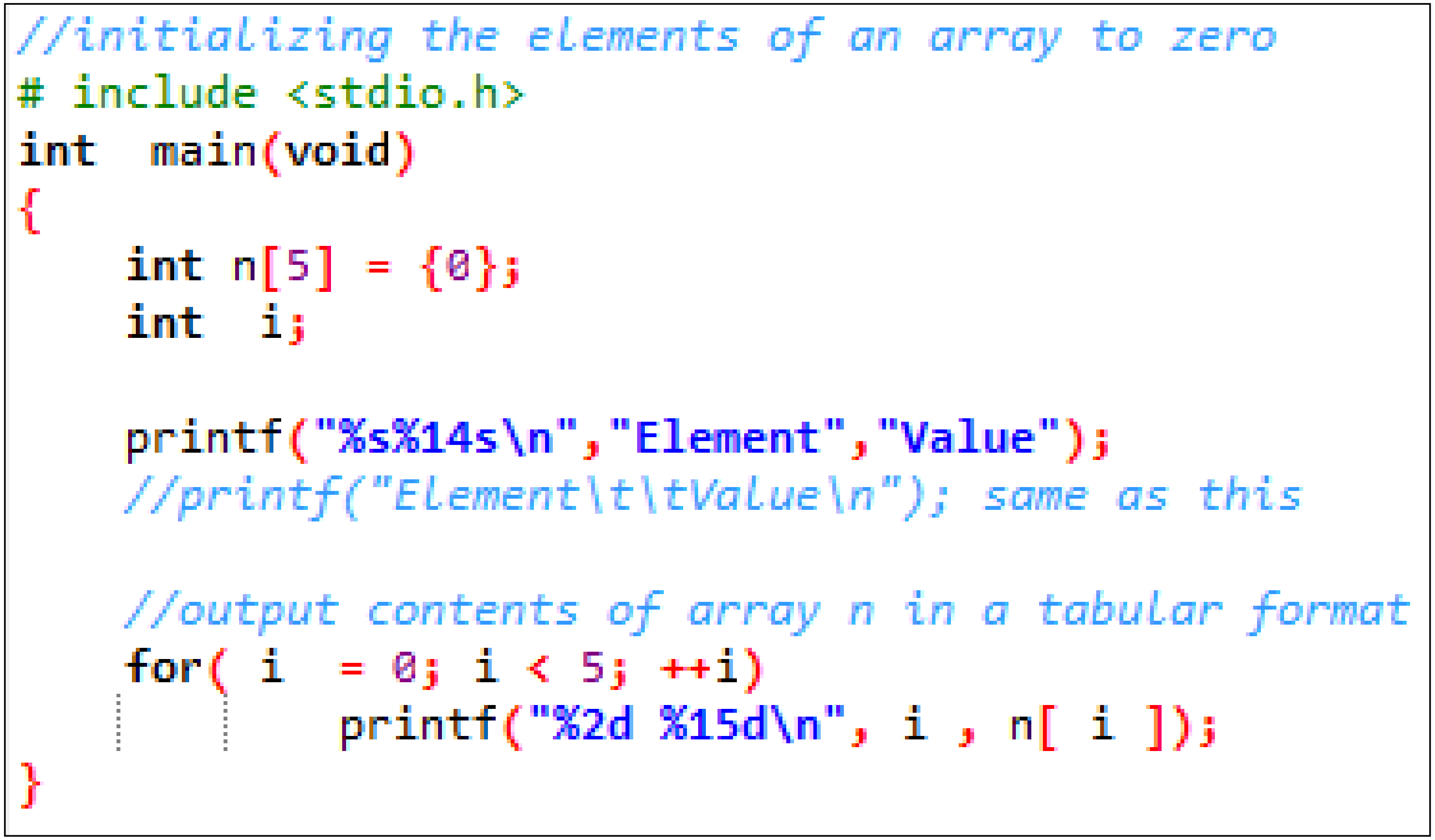
- Generally an array must be read element by element, and cannot be read all at once.

  printf(“%d ”, arr1); ❌

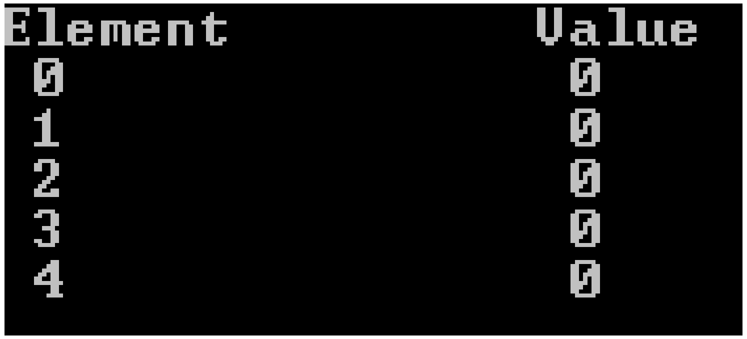
- An array cannot be assigned to another array even if they are of the same type.

  double arr1 = double arr2 ❌

- **int arr[10] = {0};** → will initialize the array to 10 zeros.



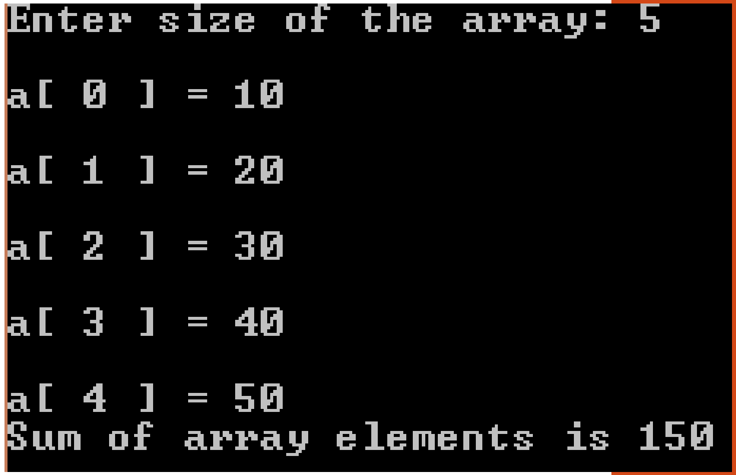
```c
//initializing the elements of an array to zero
# include <stdio.h>
int  main(void)
{
    int n[5] = {0};
    int  i;

    printf("%s%14s\n","Element","Value");
    //printf("Element\t\tValue\n"); same as this

    //output contents of array n in a tabular format
    for( i  = 0; i < 5; ++i)
            printf("%2d %15d\n", i , n[ i ]);
}
```



```
Element         Value
 0               0
 1               0
 2               0
 3               0
 4               0
```

Getting the total of array elements

```
# include <stdio.h>
void  main()
{
    int i,j,size,sum=0;

    printf("Enter size of the array: ");
    scanf("%d",&size);

    int a[ size ];

    for( i = 0; i < size; ++i){
        printf("\na[ %d ] = ",i);
        scanf("%d", &a[ i ]);
    }

    for( j = 0; j < size; ++j)
            sum += a[ j ];

    printf("Sum of array elements is %d \n", sum);
}
```



```
Enter size of the array: 5

a[ 0 ] = 10

a[ 1 ] = 20

a[ 2 ] = 30

a[ 3 ] = 40

a[ 4 ] = 50
Sum of array elements is 150
```

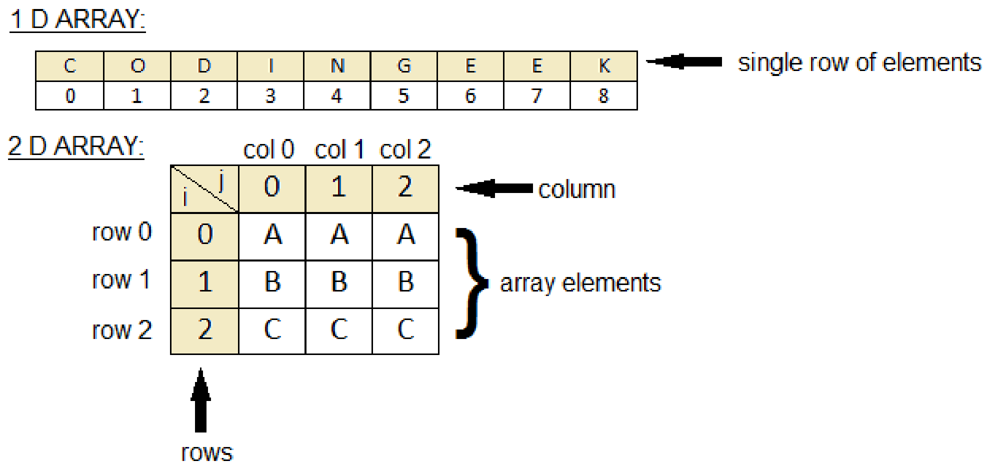
Exercise 01

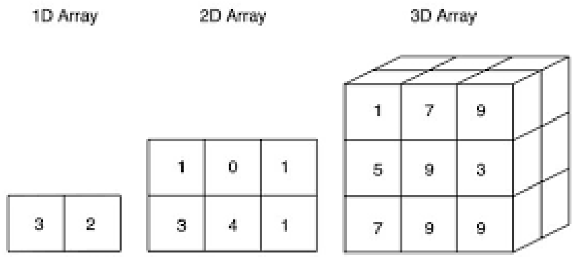
- Write a C program to do the following
  - Input 10 integer numbers between 1-10 from the user and store them in an array.
  - Then find and display the multiplication of even indexed numbers in the array.

Ex: arr[] = {**5**,**2**,**4**,**9**,**6**,**3**,**9**,**8**,**1**,**7**} → 5*4*6*9*1 →1080

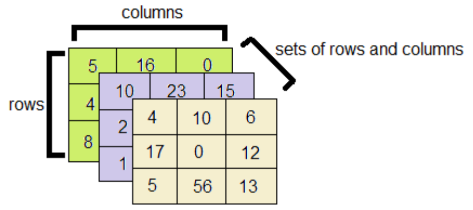
Types of Arrays

- **One dimensional arrays(1D)** – lists of values
- **Two dimensional arrays(2D)** – tables of values

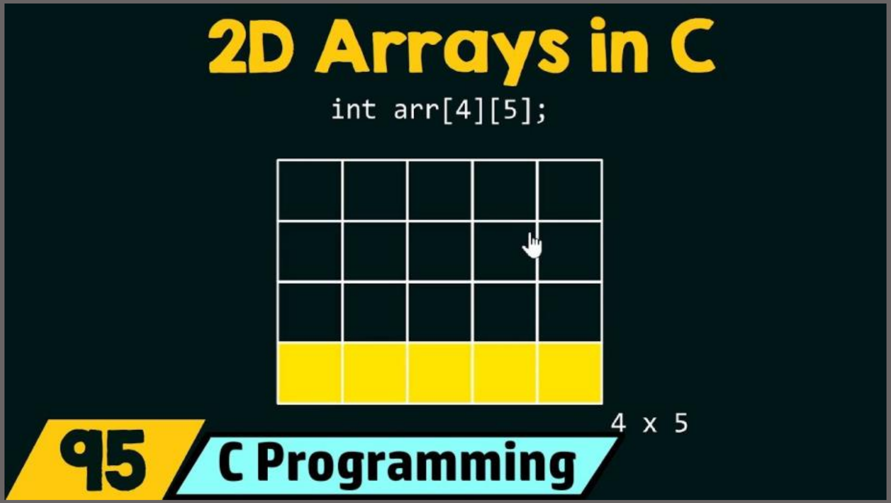




- **Higher dimension arrays(3D)** – tables of values (where the criteria for a single storage location is more than two.)



Two Dimensional Arrays



```
int arr[4][5];
```

2D Arrays

- C language has arrays with multiple subscripts.
- These arrays are refers to as multidimensional arrays.
- The simplest form of multidimensional array is the two-dimensional array which consists of rows and columns.
- It can called as an "array of arrays".

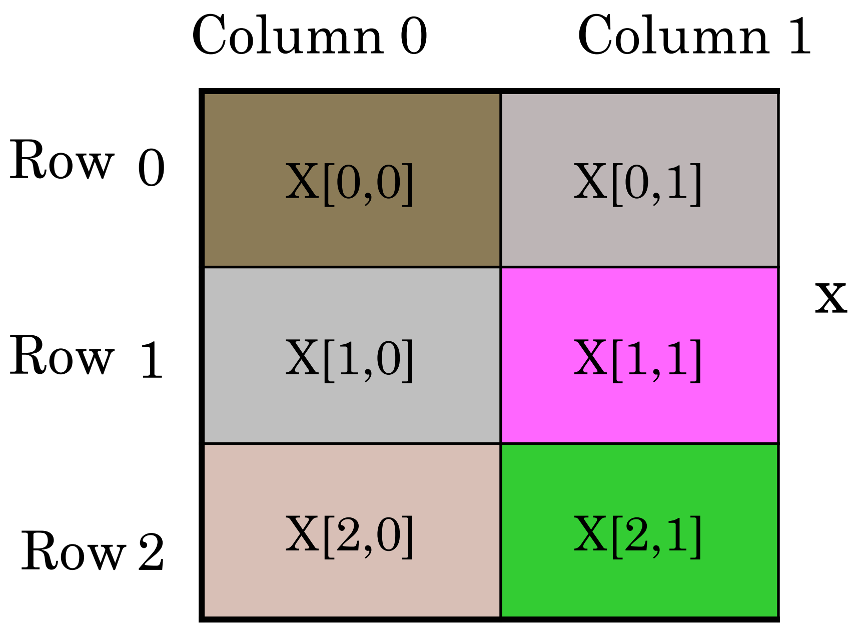
4	9	-1	5
2	4	0	8
1	7	-5	-2

2D Array Declaration

- **Syntax**:

Data_type Array_name[num_of_rows][num_of_column];

```
Ex: int x[3][2];
```



2D Array Initialization

- **Syntax**: Initialize during declaration

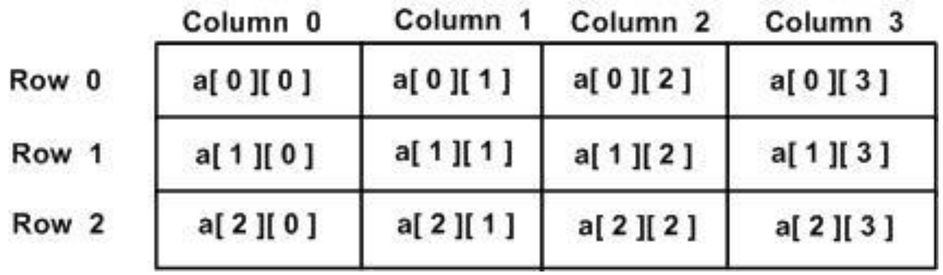
data_type arr_name[3][2] = {{v1,v2},{v3,v4},{v5,v6}}

- Example

```
     int x[3][2] = {{1,4},{6,12},{5,2}};
OR

     int x[ ][ ] = {{1,4},{6,12},{5,2}};
```

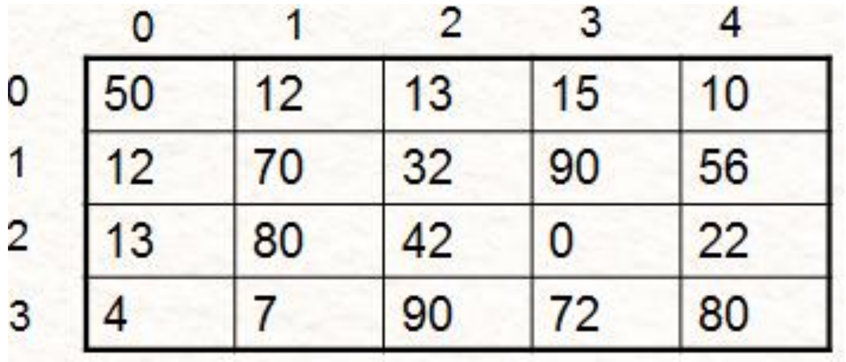
1	4
6	12
5	2

Example: Printing the values of an 2D Array



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

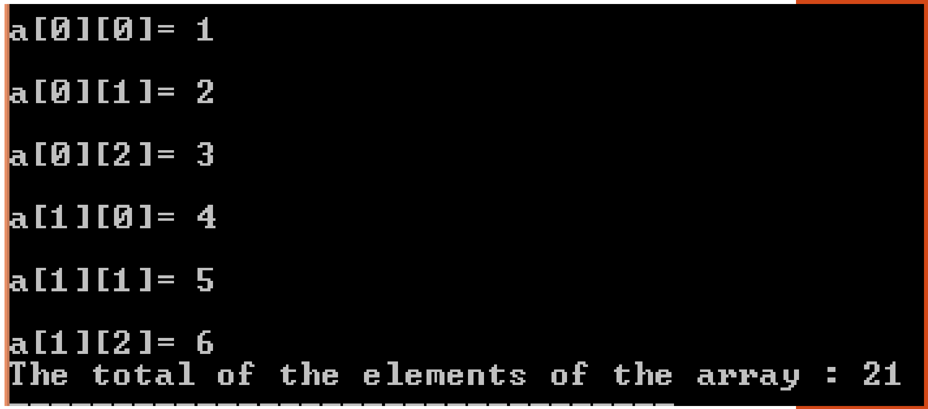
```
#include <stdio.h>
int main()
{
   int data[4][5] = {  {50, 12, 13, 15, 10}, {12, 70, 32, 90, 56},
                          {13, 80, 42, 0, 22}, {4, 7, 90, 72, 80}  };

   // Need a nested loop to go through rows and columns
   for (int r =0;  r<4;  r++)
   {
          for (int c=0;  c<5;  c++)
                   printf( "%d", data[r][c] ) ;

          printf("\n");
    }
   return 0;
}
```



	0	1	2	3	4
0	50	12	13	15	10
1	12	70	32	90	56
2	13	80	42	0	22
3	4	7	90	72	80

Example: Getting the total of Matrix values

```c
# include <stdio.h>
int  main(){
        int row,column;
        int a[2][3];
        int total = 0;

     for( row = 0; row <=1; ++row){
         for( column= 0; column<= 2; ++column)
         {
             printf("a[%d][%d]= ",row, column);
             scanf("%d",  &a[ row ][column ]);
         }
     }
     for( row = 0; row<=1; ++row){
         for( column= 0; column<=2; ++column){
             total += a[row][column];
         }
     }

     printf("The total of the elements of the array : %d", total);
     return 0;
}
```



```
a[0][0]= 1

a[0][1]= 2

a[0][2]= 3

a[1][0]= 4

a[1][1]= 5

a[1][2]= 6
The total of the elements of the array : 21
```

Exercise 02

- Create three 3 by 3 Matrices (2D-Arrays) A, B and C.
- Fill A and B with following values.
- Add Matrix A and Matrix B and store the answer in Matrix C.
- Display the content of all three Matrices.

A

1	2	3
4	5	6
7	8	9

B

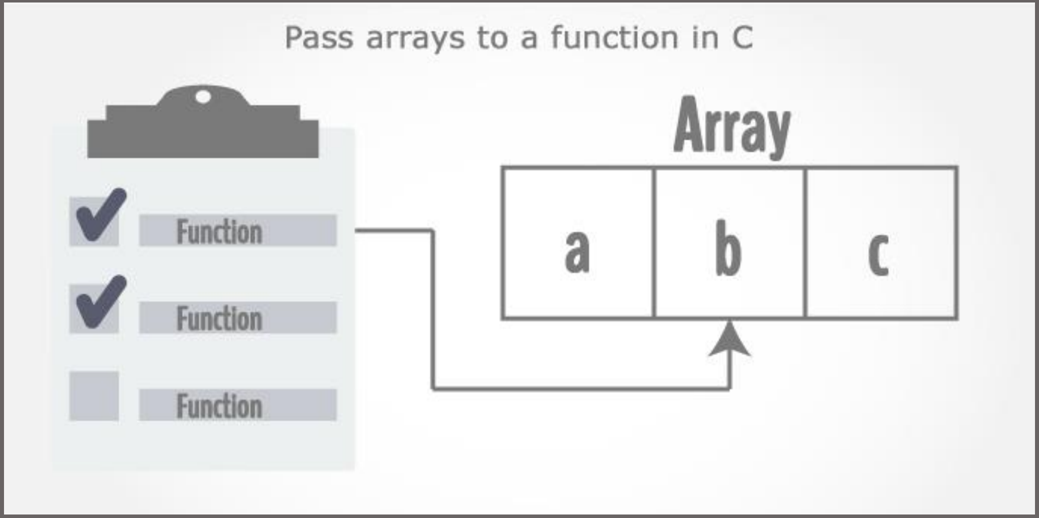
2	3	4
5	6	7
8	9	10

$\rightarrow$

C

1+2		

Arrays with Functions



Passing Arrays to Functions

- Similar as variables, arrays also can be passed into functions as parameters.
- Passing an Array into a function is an example of "Pass By Reference" in functions.

At Function Definition

- To pass 1D Array to a function, declare a formal parameter in one of following three ways.

```
1. int myFunc(int myArray[]);    3. int myFunc(int *myArray)
2. Int myFunc(int myArray[4]);
```

At Function Calling

- When passing an array as the argument at function invocation, specify only the array name without any brackets.

```
int myArray[4] = {1,5,6,12};
myFunc( myArray );
```

Returning Arrays from Functions

- Returning an Array from a function is not simple as returning a variable.
- Functions can not return an entire array.
- But it can return a pointer to an array by specifying the array name without an index.

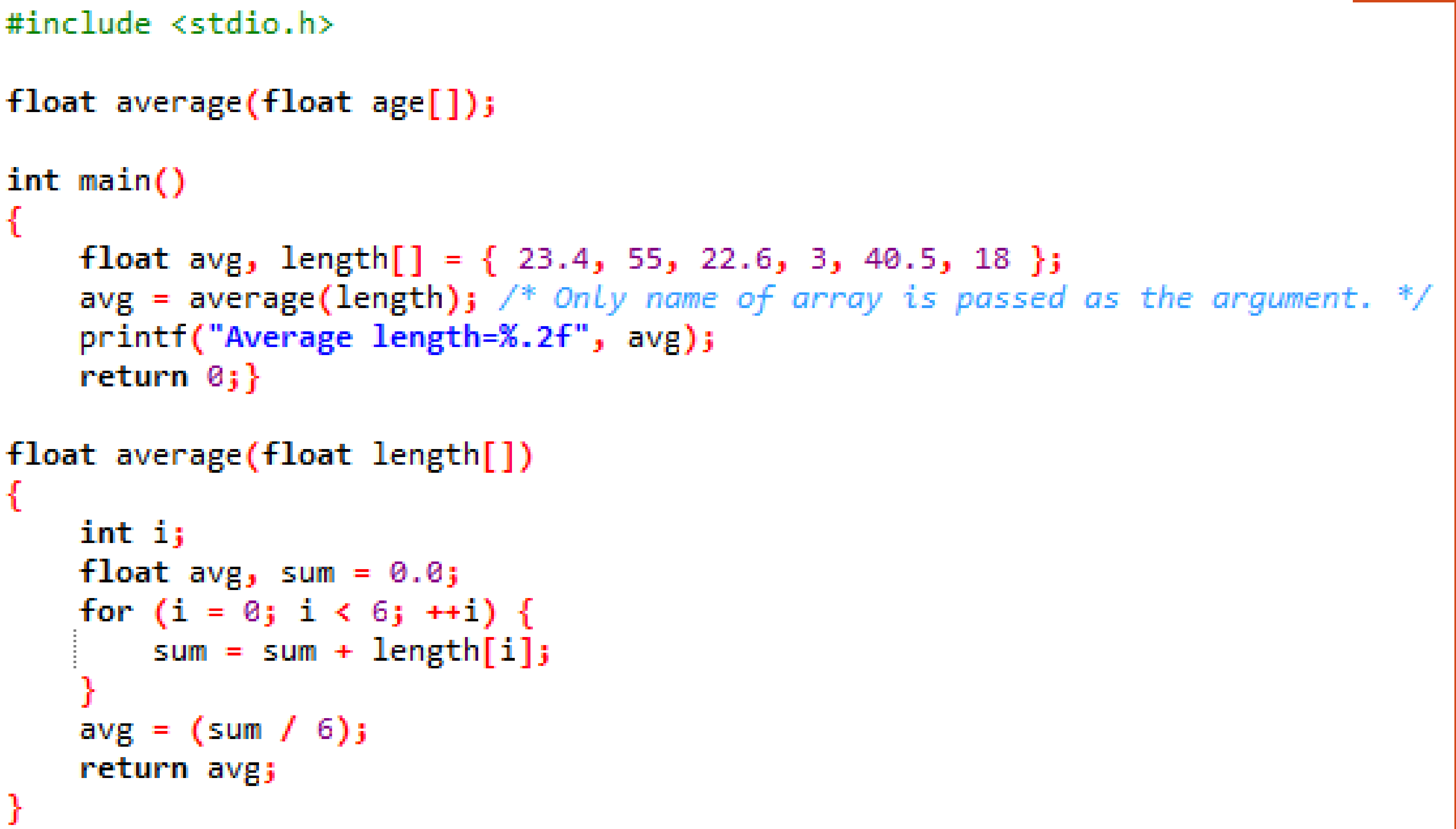
```
int *myFunction(int myArray[]){

        return myArray;
}
```

- To return a local array (array declared within the function), array should define as static in C.

```
int *myFunction( ){

        static int myArray[5]; ............

        return myArray;
}
```

Example

Write a program to pass an array of length of boxes to a function. Function should find and display the average length.

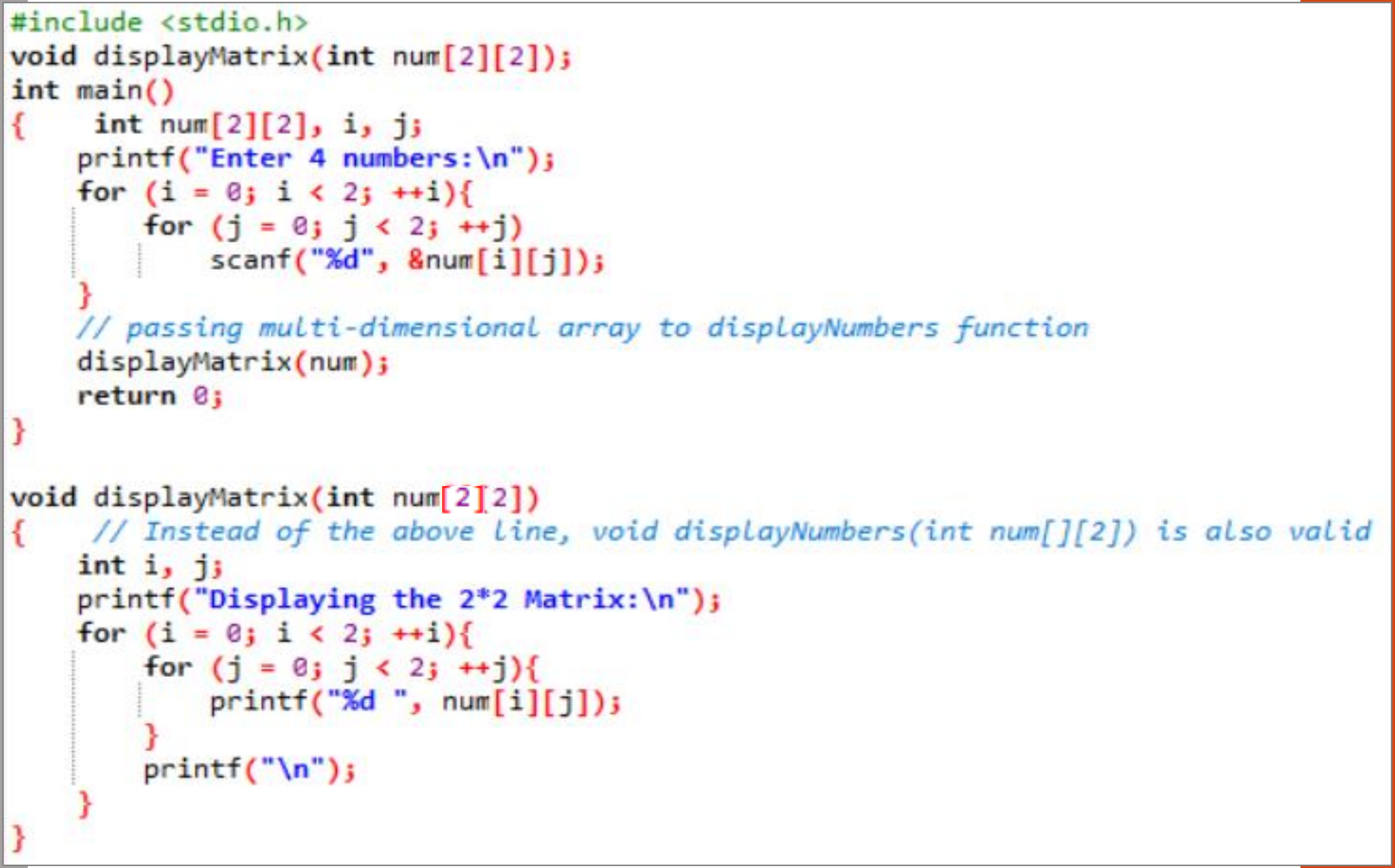


```c
#include <stdio.h>

float average(float age[]);

int main()
{
    float avg, length[] = { 23.4, 55, 22.6, 3, 40.5, 18 };
    avg = average(length); /* Only name of array is passed as the argument. */
    printf("Average length=%.2f", avg);
    return 0;}

float average(float length[])
{
    int i;
    float avg, sum = 0.0;
    for (i = 0; i < 6; ++i) {
        sum = sum + length[i];
    }
    avg = (sum / 6);
    return avg;
}
```

```
# include <stdio.h>
# define SIZE 5
void modifyArray(int b[ ], int size);

int main ( void ){

          int a[SIZE]  = { 0, 1, 2, 3, 4};
          modifyArray(a, SIZE);

   return 0;
}

void modifyArray( int b [  ], int size)
{
          int  j;
          for( j  = 0; j < size; ++j){
                    b[j] *= 2; // multiply each array element by 2
                    printf("%d ",b[j]);
          }
}
```

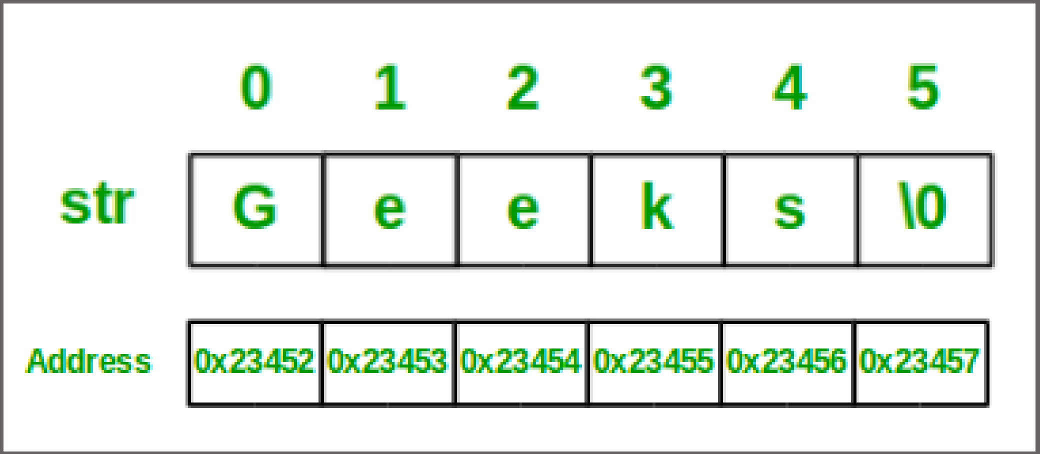
```
0 2 4 6 8
```

Passing 2D Array to a function



```c
#include <stdio.h>
void displayMatrix(int num[2][2]);
int main()
{    int num[2][2], i, j;
    printf("Enter 4 numbers:\n");
    for (i = 0; i < 2; ++i){
        for (j = 0; j < 2; ++j)
            scanf("%d", &num[i][j]);
    }
    // passing multi-dimensional array to displayNumbers function
    displayMatrix(num);
    return 0;
}

void displayMatrix(int num[2][2])
{    // Instead of the above line, void displayNumbers(int num[][2]) is also valid
    int i, j;
    printf("Displaying the 2*2 Matrix:\n");
    for (i = 0; i < 2; ++i){
        for (j = 0; j < 2; ++j){
            printf("%d ", num[i][j]);
        }
        printf("\n");
    }
}
```

Strings



0
1
2
3
4
5
str
G
e
e
k
s
\0
Address
0x23452
0x23453
0x23454
0x23455
0x23456
0x23457

String

- A **string** is a series of characters treated as a single unit.
- A string may include letters, digits and various **special characters** written within double quotation marks.
- C does not have a string data type.
- A string in C is an **array of characters** **(1-D)** ending with the null character ('\0').

```
char  myString [ ] = “Orange”;
```

```
char  myString[ ] = {‘O’,’r’,’a’,’n’,’g’,’e’,‘\0’};
```

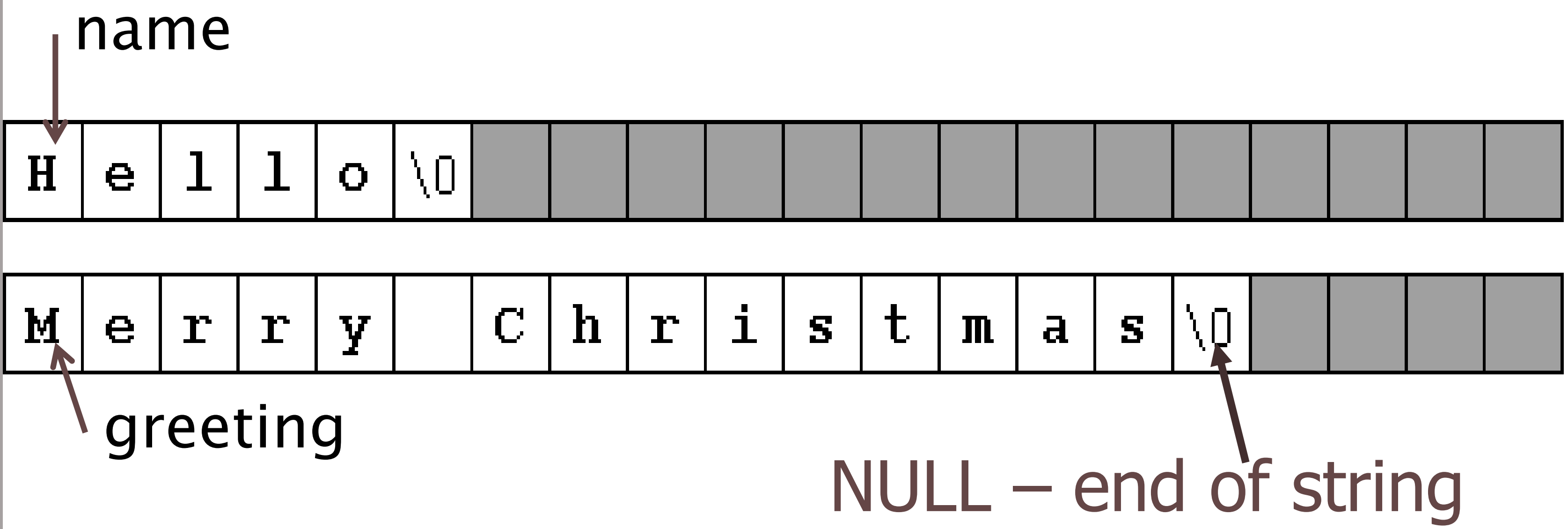
The easiest way to have string data is using double quotations. Do not want to specify null character.

String Handling

- String is accessed via a *pointer* to the first character in the string.

name of the array ← address of the 1st element of the array

- And the null character indicates the end of the string.
- A string can be stored in a character array as follows.



String with scanf( )

A string can be stored in an array using scanf. Ex:

1. scanf with a char

```
char word[5];
for(int i=0;i<5;i++)
        scanf( "   %c" , &word[i] );
```

2. scanf with a string

```
scanf( "%s", word );
```

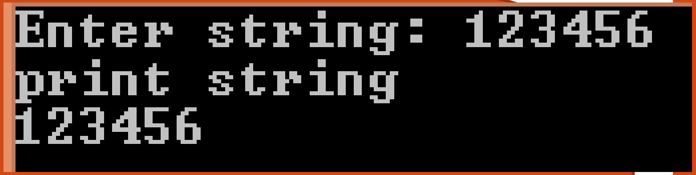
'&' is used to get the address of the variable.

The string **array name** itself **points to the base address** (address of the 1st element) of the array and therefore **no need of adding an extra '&'.**

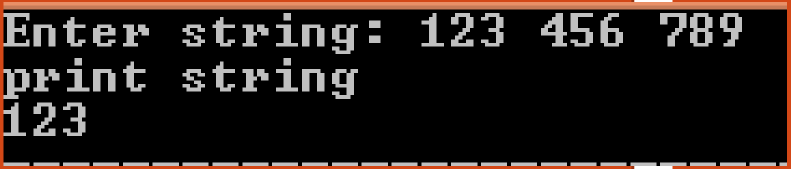
String with scanf( )

- Function scanf() will read characters until **space**, **tab**, **newline** or **end-of-file indicator** is encountered.

```
#include<stdio.h>
void main(){
	char text[10];
	int i;

	printf("Enter string: ");
	scanf("%s",text);

	printf("print string\n");
	printf("%s",text); //print as a char array
}
```



```
Enter string: 123456
print string
123456
```



```
Enter string: 123 456 789
print string
123
```

Printing strings

```
#include<stdio.h>
int main(){
      char text[10]="Hi World";
      int i;

      printf("%s",text); //print directly as a
                         //string

      for(i=0;i<10;i++){
            printf("%c",text[i]); //print as a
                                  //char array
      }

return 0;
}
```

The header file <string.h>

Some functions in the <string.h> header file are shown below.

strlen(S1) : Returns the length of a string.

strcpy(S1,S2) : Copies string S2 into string S1. The value of S1 is returned.

strcmp(S1,S2) : Compares S1 with S2. The function returns 0,less than 0 or greater than 0 if S1 is equal to, less than or greater than S2, respectively.

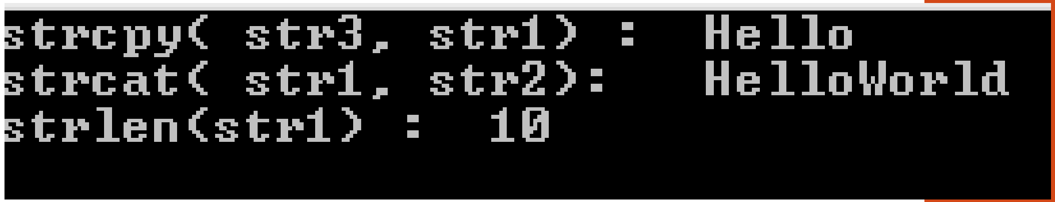
strcat(S1, S2) : Concatenates string S2 onto the end of string s1. The first character of S2 overwrites the terminating null character of s1. The value of S1 is returned.

strcmp( ) in <string.h>

```
char s1[20], s2[20];
int status;
printf( "Enter Name 1 : “);
scanf(“%s”, s1);
printf ("Enter Name 2 : “);
scanf(“%s”, s2);
status = strcmp(s1, s2);
```

strcmp(s1,s2) - Compare two string variables. It returns,
0 if the strings are equal.
>0 if s1 is after s2 alphabetically.
<0 if s1 is before s2 alphabetically.

```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[12] = "Hello";
    char str2[12] = "World";
    char str3[12];
    int  len ;

    /* copy str1 into str3 */
    strcpy(str3, str1);
    printf("strcpy( str3, str1) :  %s\n", str3 );

    /* concatenates str1 and str2 */
    strcat( str1, str2);
    printf("strcat( str1, str2):   %s\n", str1 );

    /* total lenghth of str1 after concatenation */
    len = strlen(str1);
    printf("strlen(str1) :  %d\n", len );
    return 0;
}
```

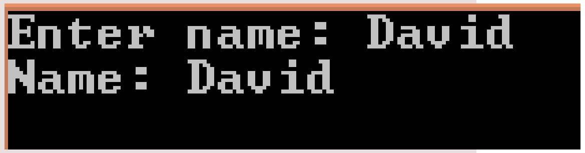


```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

gets( ) and puts( ) string functions

Functions gets() and puts() handle strings, both these functions are defined in "stdio.h" header file.

```
#include<stdio.h>
int main(){
    char name[30];

    printf("Enter name: ");
    gets(name);     //Function to read string from user.

    printf("Name: ");
    puts(name);     //Function to display string.
    return 0;
}
```



```
Enter name: David
Name: David
```

Any
Question

THANK YOU… !