CS101: Problem Solving through C Programming

Sachchida Nand Chaurasia

Assistant Professor

Department of Computer Science
Banaras Hindu University
Varanasi
Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

February 22, 2021

Arrays I

✓ The variable allows us to store a single value at a time, what if we want to store roll no. of 100 students?

✓ For this task, we have to declare 100 variables, then assign values to each of them.

✓ What if there are 10000 students or more?

✓ As you can see declaring that many variables for a single entity (i.e student) is not a good idea.

✓ In a situation like these arrays provide a better way to store data.

Arrays II

What is an Array?

✓ An array is a collection of one or more values of the same type.

✓ Each value is called an element of the array.

✓ The elements of the array share the same variable name but each element has its own unique index number (also known as a subscript).

✓ An array can be of any type, For example: int, float, char etc. If an array is of type int then it's elements must be of type int only.

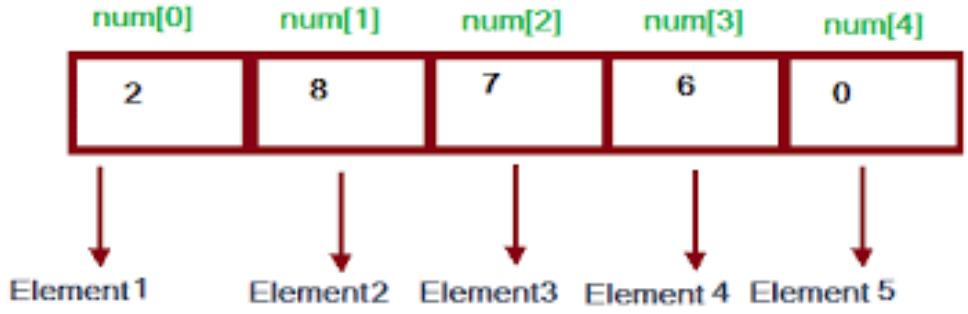
Arrays III

Arrays IV

To store roll no. of 100 students, we have to declare an array of size 100 i.e roll_no[100]. Here size of the array is 100 , so it is capable of storing 100 values. In C, index or subscript starts from 0, so roll_no[0] is the first element, roll_no[1] is the second element and so on. Note that the last element of the array will be at roll_no[99] not at roll_no[100] because the index starts at 0.

Arrays V

One-dimensional array:




```
Syntax: datatype array_name[size];
datatype: It denotes the type of the elements in the array.
array_name: Name of the array. It must be a valid identifier.
size: Number of elements an array can hold. here are some example of array declarations:

int num[100];
float temp[20];
char ch[50];

num[0], num[1], num[2], ....., num[99]
temp[0], temp[1], temp[2], ....., temp[19]
ch[0], ch[1], ch[2], ....., ch[49]
```

Arrays VI

```c
#define SIZE 10
int main()
{
	int size = 10;

	int my_arr1[SIZE]; // ok
	int my_arr1[10]; // ok
	int my_arr2[size]; // not allowed
	// ...
}
```

Accessing elements of an array:

✓ The elements of an array can be accessed by specifying array name followed by subscript or index inside square brackets (i.e []).

✓ Array subscript or index starts at 0. If the size of an array is 10 then the first element is at index 0, while the last element is at index 9.

✓ The first valid subscript (i.e 0) is known as the lower bound, while last valid subscript is known as the upper bound.

Arrays VII

```c
int my_arr[5];

//then elements of this array are;

First element  -> my_arr[0]
Second element -> my_arr[1]
Third element  -> my_arr[2]
Fourth element -> my_arr[3]
Fifth element  -> my_arr[4]

//Array subscript or index can be any expression that yields an integer value. For example:

int i = 0, j = 2;
my_arr[i]; // 1st element
my_arr[i+1]; // 2nd element
my_arr[i+j]; // 3rd element

//In the array my_arr, the last element is at my_arr[4], What if you try to access elements beyond the last
      valid index of the array?

printf("%d", my_arr[5]); // 6th element
printf("%d", my_arr[10]); // 11th element
printf("%d", my_arr[-1]); // element just before 0

```

Arrays VIII

```
24 //Sure indexes 5, 10 and -1 are not valid but C compiler will not show any error message instead some garbage
       value will be printed.
25 //The C language doesn't check bounds of the array. It is the responsibility of the programmer to check array
       bounds whenever required.
```

Processing 1-D arrays:

```
1  #include<stdio.h>
2  int main()
3  {
4         int arr[5], i;
5         for(i = 0; i < 5; i++)
6         {
7               printf("Enter a[%d]: ", i);
8               scanf("%d", &arr[i]);
9         }
10        printf("\nPrinting elements of the array: \n\n");
11        for(i = 0; i < 5; i++)
12        {
13              printf("%d ", arr[i]);
14        }
15        return 0;
16 }
```

Arrays IX

```c
//The following program prints the sum of elements of an array.
#include<stdio.h>
int main()
{
	int arr[5], i, s = 0;
	for(i = 0; i < 5; i++)
	{
		printf("Enter a[%d]: ", i);
		scanf("%d", &arr[i]);
	}
	for(i = 0; i < 5; i++)
	{
		s += arr[i];
	}
	printf("\nSum of elements = %d ", s);
	return 0;
}
```

Arrays X

Initializing Array:

✓ When an array is declared inside a function the elements of the array have garbage value.

✓ If an array is global or static, then its elements are automatically initialized to 0.

✓ We can explicitly initialize elements of an array at the time of declaration using the following syntax:

Syntax:

$\Longrightarrow$ datatype array_name[size] ={val_1, val_2, val_3, ... val_N};

$\Longrightarrow$ datatype is the type of elements of an array.

$\Longrightarrow$ array_name is the variable name, which must be any valid identifier.

$\Longrightarrow$ size is the size of the array.

✓ val_1, val_2, val_3, ... val_N are the constants known as initializers.

Arrays XI

Tick Each value is separated by a comma(,) and then there is a semi-colon (;) after the closing curly brace (}).

```
float temp[5] =
{
      12.3, 4.1, 3.8, 9.5, 4.5
}
; // an array of 5 floats
int arr[9] =
{
      11, 22, 33, 44, 55, 66, 77, 88, 99
}
; // an array of 9 ints
```

Arrays XII

✓ While initializing 1-D array it is optional to specify the size of the array, so you can also write the above statements as:

```
float temp[] =
{
      12.3, 4.1, 3.8, 9.5, 4.5
}
; // an array of 5 floats
int arr[] =
{
      11, 22, 33, 44, 55, 66, 77, 88, 99
}
; // an array of 9 ints
```

Arrays XIII

```c
#include<stdio.h>
#define SIZE 10

int main()
{
	int my_arr[SIZE] =
	{
		34,56,78,15,43,71,89,34,70,91
	}
	;
	int i, max, min;

	max = min = my_arr[0];

	for(i = 0; i < SIZE; i++)
	{
		// if value of current element is greater than previous value
		// then assign new value to max
		if(my_arr[i] > max)
		{
			max = my_arr[i];
		}

		// if the value of current element is less than previous element
		// then assign new value to min
```

Arrays XIV

```
26              if(my_arr[i] < min)
27              {
28                      min = my_arr[i];
29              }
30
31      }
32
33      printf("Lowest value = %d\n", min);
34      printf("Highest value = %d", max);
35
36      // signal to operating system everything works fine
37      return 0;
38  }
```

Arrays XV

```c
// Program to find the average of n numbers using arrays
#include <stdio.h>
int main()
{
        int marks[10], i, n, sum = 0, average;
        printf("Enter number of elements: ");
        scanf("%d", &n);

        for(i=0; i<n; ++i)

      {
              printf("Enter number%d: ",i+1);
              scanf("%d", &marks[i]);

              // adding integers entered by the user to the sum variable
              sum += marks[i];

      }
        average = sum/n;
        printf("Average = %d", average);

        return 0;
}
```

Practice questions on 1-D Arrays

Type 1. Based on array declaration – These are few key points on array declaration:

A single dimensional array can be declared as int a[10] or int a[] = {1, 2, 3, 4}. It means specifying the number of elements is optional in 1-D array. A two dimensional array can be declared as int a[2][4] or int a[][4] = {1, 2, 3, 4, 5, 6, 7, 8}. It means specifying the number of rows is optional but columns are mandatory. The declaration of int a[4] will give the values as garbage if printed. However, int a[4] = 1,1 will initialize remaining two elements as 0.

Arrays XVII

```
1  int main()
2  {
3          int i;
4          int arr[5] =
5          {
6                  1
7          }
8          ;
9          for (i = 0; i < 5; i++)
10         printf("%d ", arr[i]);
11         return 0;
12 }
13 Output: ???
```

Arrays XVIII

Type 2. Finding address of an element with given base address - When an array is declared, a contiguous block of memory is assigned to it which helps in finding address of elements from base address.
For a single dimensional array a[100], address of ith element can be found as:
addr(a[i]) = BA+ i*SIZE

Arrays XIX

Type 3. Accessing array elements using pointers -

In a single dimensional array a[100], the element a[i] can be accessed as a[i] or *(a+i) or *(i+a) Address of a[i] can be accessed as &a[i] or (a+i) or (i+a) In two dimensional array a[100][100], the element a[i][j] can be accessed as a[i][j] or *(*(a+i)+j) or *(a[i]+j) Address of a[i][j] can be accessed as &a[i][j] or a[i]+j or *(a+i)+j In two dimensional array, address of ith row can be accessed as a[i] or *(a+i)

Arrays XX

```
Assume the following C variable declaration

int *A [10], B[10][10];
Of the following expressions

I. A[2]
II. A[2][3]
III. B[1]
IV. B[2][3]
which will not give compile-time errors if used as left hand sides of assignment statements in a C program (
      GATE CS 2003)?
(A) I, II, and IV only
(B) II, III, and IV only
(C) II and IV only
(D) IV only

Solution: As given in the question, A is an array of 10 pointers and B is a two dimensional array. Considering
       this, we take an example as:

int *A[10], B[10][10];
int C[] =
{
      1, 2, 3, 4, 5
}
;
```

Arrays XXI

```
24 As A[2] represents an integer pointer, it can store the address of integer array as: A[2] = C; therefore, I is
       valid.
25
26 As A[2] represents base address of C, A[2][3] can be modified as: A[2][3] = *(C +3) = 0; it will change the
      value of C[3] to 0. Hence, II is also valid.
27
28 As B is 2D array, B[2][3] can be modified as: B[2][3] = 5; it will change the value of B[2][3] to 5. Hence, IV
       is also valid.
29
30 As B is 2D array, B[2] represent address of 2nd row which can't be used at LHS of statement as it is invalid
      to modify the address. Hence III is invalid.
```

2-D arrays:

✓ The syntax declaration of 2-D array is not much different from 1-D array. In 2-D array, to declare and access elements of a 2-D array we use 2 subscripts instead of 1.

✓ Syntax: datatype array_name[ROW][COL];

✓ The total number of elements in a 2-D array is ROW*COL. Let's take an example.

```
1 int arr[3][3];
```

Arrays XXIII

✓ The individual elements of the above array can be accessed by using two subscript instead of one.

✓ The first subscript denotes row number and second denotes column number. ✓ As we can see in the above image both rows and columns are indexed from 0. So the first element of this array is at arr[0][0] and the last element is at arr[1][2]. Here are how you can access all the other elements:

arr[0][0] - refers to the first element

arr[0][1] - refers to the second element

arr[0][2] - refers to the third element

arr[1][0] - refers to the fourth element

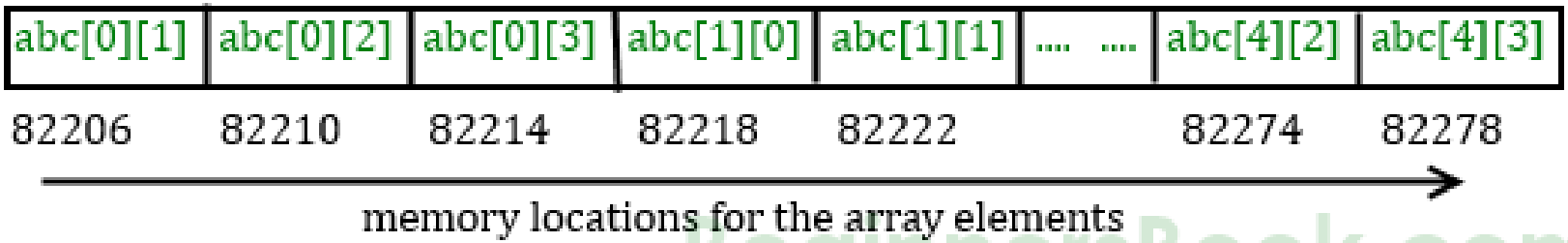
arr[1][1] - refers to the fifth element

arr[1][2] - refers to the sixth element

Arrays XXIV

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Arrays XXV

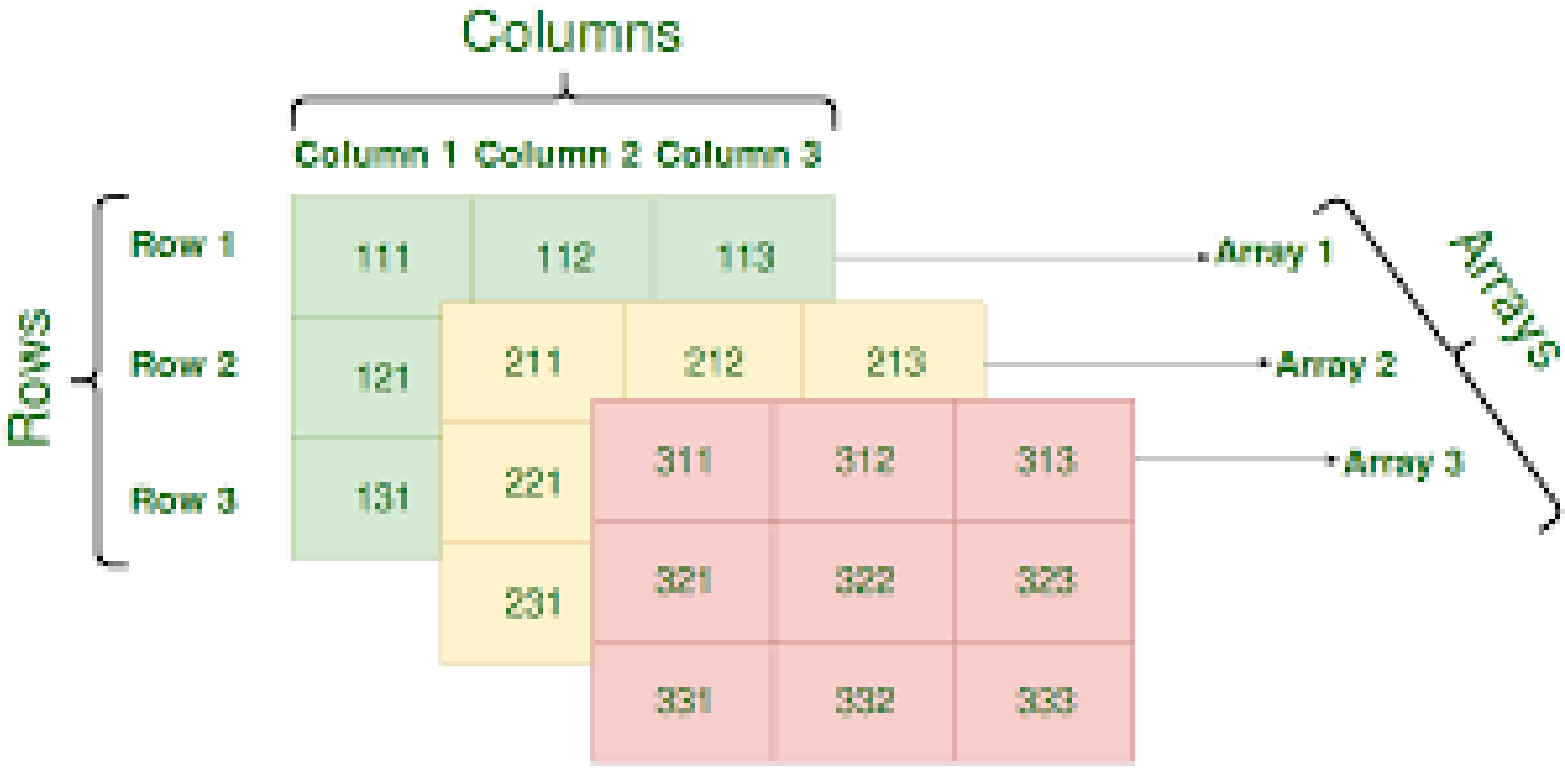


Array is of integer type so each element would use 4 bytes that's the reason there is a difference of 4 in element's addresses.

The addresses are generally represented in hex. This diagram shows them in integer just to show you that the elements are stored in contiguos locations, so that you can understand that the address difference between each element is equal to the size of one element(int size 4). For better understanding see the program below.

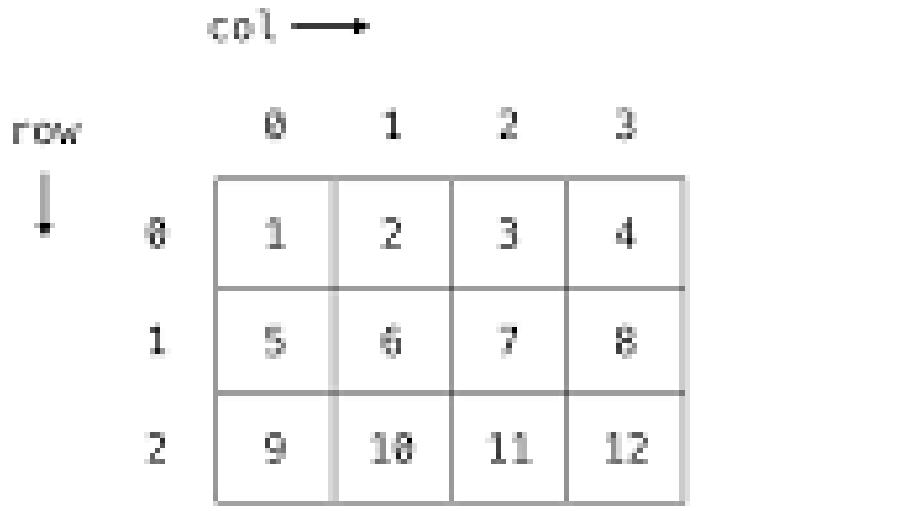
Actual memory representation of a 2D array

Arrays XXVI


Columns
Column 1 Column 2 Column 3
Rows
Row 1
Row 2
Row 3
111
112
113
121
211
212
213
131
221
231
311
312
313
321
322
323
331
332
333
Array 1
Array 2
Array 3
Arrays

Arrays XXVII

```
int num[3][4] = {
  {1,  2,  3,  4},
  {5,  6,  7,  8},
  {9, 10, 11, 12}
};
```



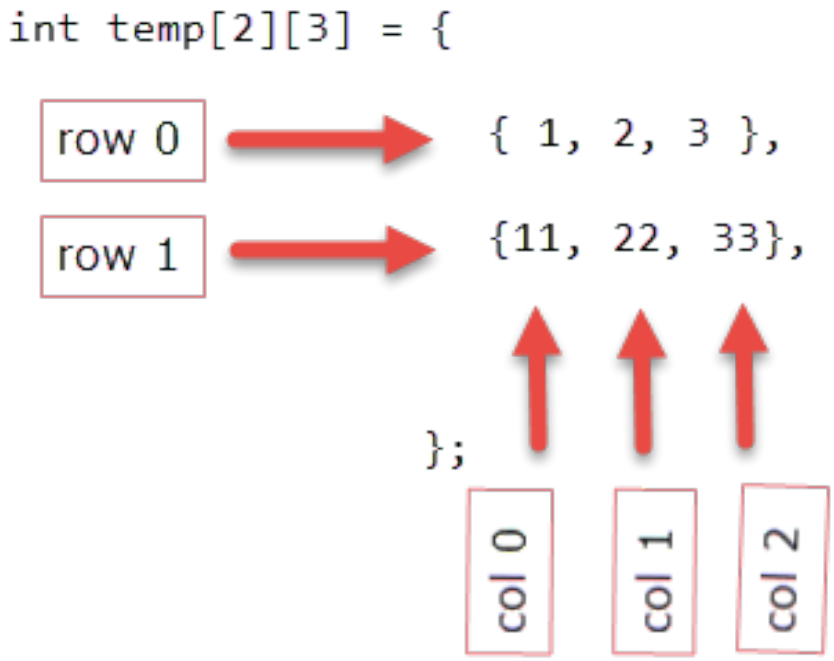

```
1 #define ROW 2
2 #define COL 3
3
4 int i = 4, j = 6;
5 int arr[ROW][COL]; // OK
6 int new_arr[i][j]; // ERROR
```

Arrays XXVIII

Initializing 2-D array:

```
int temp[2][3] =
{
       {
              1, 2, 3
       }
       , // row 0
       {
             11, 22, 33
       }
        // row 1
}
;
```

Arrays XXIX



Arrays XXX

```
int my_arr[4][3] = {
{10},
{77, 92},
{33, 89, 44},
{12, 11}
};
```

```
my_arr[0][0] : 10
my_arr[0][1] : 0
my_arr[0][2] : 0

my_arr[1][0] : 77
my_arr[1][1] : 92
my_arr[1][2] : 0

my_arr[2][0] : 33
my_arr[2][1] : 89
my_arr[2][2] : 44

my_arr[3][0] : 12
```

Arrays XXXI

```
14 my_arr[3][1] : 11
15 my_arr[4][2] : 0
```

int two_d[][3] = {
{13,23,34},
{15,27,35}
};
is same as
int two_d[2][3] = {
{13, 23, 34},
{15, 27, 35}
};

Practice questions on 2-D Arrays

```
1  int main()
2  {
3         int a[][] =
4         {
5
6                 {
7                         1,2
8                 }
9                 ,
10                {
11                        3,4
12                }
13
14        }
15        ;
16        int i, j;
17        for (i = 0; i < 2; i++)
18        for (j = 0; j < 2; j++)
19        printf("%d ", a[i][j]);
20        return 0;
21 }
```

✓ Consider the following declaration of a ‘two-dimensional array in C:

Arrays XXXIII

```
char a[100][100];
Assuming that the main memory is byte-addressable and that the array is stored starting from memory address
       0, the address of a[40][50] is: (GATE CS 2002)
(A) 4040
(B) 4050
(C) 5040
(C) 5050

Solution:
addr[40][50] = 0 + (40*100 + 50) * 1 = 4050
```

✓ For a C program accessing X[i][j][k], the following intermediate code is generated by a compiler. Assume that the size of an integer is 32 bits and the size of a character is 8 bits. (GATE-CS-2014)

Arrays XXXIV

```
Que - 4. For a C program accessing X[i][j][k], the following intermediate code is generated by a compiler.
       Assume that the size of an integer is 32 bits and the size of a character is 8 bits. (GATE-CS-2014)

t0 = i * 1024
t1= j * 32
t2 = k * 4
t3 =t1 + t0
t4 = t3 + t2
t5 = X[t4]
Which one of the following statement about the source code of C program is correct?
(A) X is declared as "int X[32][32][8]"
(B) X is declared as "int X[4][1024][32]"
(C) X is declared as "char X[4][32][8]"
(D) X is declared as "char X[32][16][2]"

Solution: For a three dimensional array X[10][20][30], we have 10 two dimensional matrices of size [20]*[30].
       Therefore, for a 3 D array X[M][N][O], the address of X[i][j][k] can be calculated as:

BA + (i*N*O+j*O+k)*SIZE
Given different expressions, the final value of t5 can be calculated as:




t5 = X[t4] = X[t3+t2] = X[t1+t0+t2] = X[i*1024+j*32+k*4]
By equating addresses,
```

Arrays XXXV

```
24
25 (i*N*O+j*O+k)SIZE = i*1024+j*32+k*4 = (i*256+j*8+k)4
26 Comparing the values of i, j and SIZE, we get
27
28 SIZE = 4, N*O = 256 and O = 8, hence, N = 32
29 As size is 4, array will be integer. The option which matches value of N and O and array as integer is (A).
```

Arrays of more than two dimension

✓ You can even create an array of 3 or more dimensions or more, but generally, you will never need to do so. Therefore, we will restrict ourself to 3-D arrays only.

✓ Here is how you can declare an array of 3 dimensions.

```
1 int arr[2][3][2];
```

✓ 3-D array uses three indexes or subscript. This array can store 2*3*2=12 elements.

Here is how to initialize a 3-D array.

int three_d[2][3][4] = {

{

{12,34,56,12},

Arrays XXXVII

```
{57,44,62,14},
{64,36,91,16},
},
{
{87,11,42,82},
{93,44,12,99},
{96,34,33,26},
}
};
```

String I

In C programming, a string is a sequence of characters terminated with a null character

0. For example:

```
1 char c[] = "c string";
```

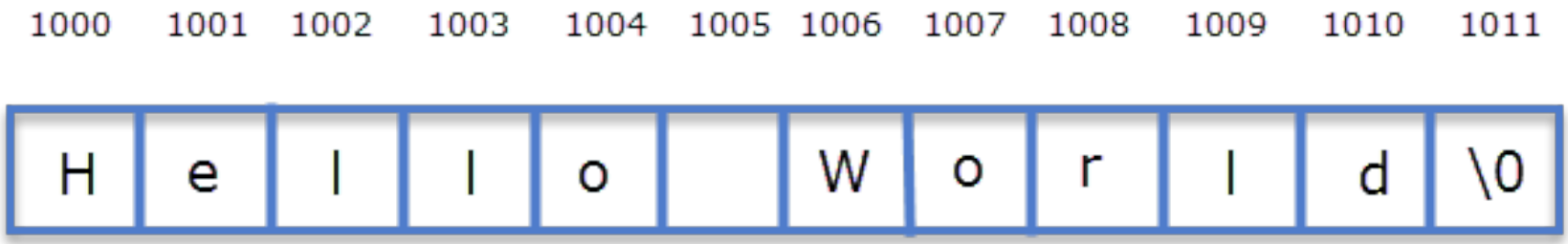
```
1  char c[] = "abcd";
2  char c[50] = "abcd";
3  char c[] =
4  {
5        'a', 'b', 'c', 'd', $'\setminus 0'$
6  }
7  ;
8  char c[5] =
9  {
10       'a', 'b', 'c', 'd', $'\setminus 0'$
11 }
12 ;
```

String literal

String II

```
1 "I am learning C"
2 "My Lucky Number is 1"
3 "Hello World!"
4 ""
```

How string literals are stored ?



String III

String literal as a Pointer:

```
1 char *str = "Hello World";
2 printf("%c" ,*(str+0) ); // prints H
3 printf("%c" ,*(str+4) ); // prints o
4
5
6 *str = 'Y'; // wrong
```

printf() and scanf() revisited:

```
1 int printf (const char*, ...);
2 int scanf (const char*, ...);
```

✓ If you look at the prototype of scanf() and print(), you will find that both function expects a value of type (char*) as their first argument.

```
1 printf("Hello World");
```

String IV

✓ You are actually passing an address of "Hello World" i.e a pointer to the first letter of the array which is 'H'.

String literal v/s character literal

Beginners often confuse between "a" and 'a', the former is a string literal where "a" is a pointer to the memory location which contains the character 'a' followed by a null character ($'\backslash 0'$).

✓ On the other hand character literal, 'a' represents the ASCII value of the character 'a' which is 97. Therefore you must never use character literal where a string literal is required or vice versa.

Multiline strings literals:

```
1 printf("This is first line \
2 some characters in the second line \
3 even more characters in the third line \n");
```

String V

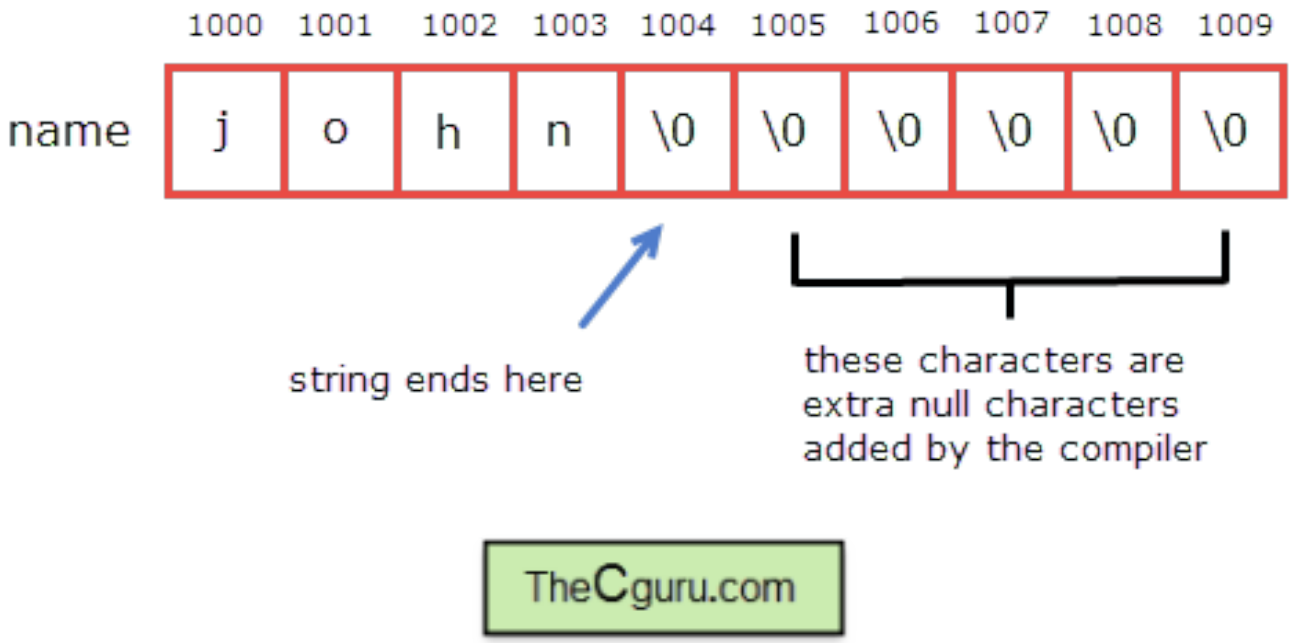
String Variables:

```
1 char ch_arr[6];
2 char ch_arr[] =
3 {
4       'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'
5 }
6 ;
7 char ch_arr[] = "Hello World";
```

✓ What if the number of characters(including $'\backslash 0'$) to be stored is less than the size of the array. In that case, the compiler adds extra null characters ($'\backslash 0'$). For example:

```
1 char name[10] = "john";
```

String VI



The strlen() Function:

```
1 size_t strlen (const char* str);
```

String VII

```
strlen("a string constant"); // returns 17

char arr[] = "an array of characters";
strlen(arr); // returns 22
```

✓ Create our own version of the strlen() function.

```
unsigned int my_strlen(char *p)
{
	unsigned int count = 0;

	while(*p!='\0')
	{
		count++;
		p++;
	}

	return count;
}
```

The strcmp() Function in C:

String VIII

```
int strcmp (const char* str1, const char* str2);

Example:

strcmp("a", "a"); // returns 0 as ASCII value of "a" and "a" are same i.e 97
strcmp("a", "b"); // returns -1 as ASCII value of "a" (97) is less than "b" (98)
strcmp("a", "c"); // returns -1 as ASCII value of "a" (97) is less than "c" (99)
strcmp("z", "d"); // returns 1 as ASCII value of "z" (122) is greater than "d" (100)
strcmp("abc", "abe"); // returns -1 as ASCII value of "c" (99) is less than "e" (101)
strcmp("apples", "apple"); // returns 1 as ASCII value of "s" (115) is greater than "\0" (101)
```

✓ The following program compares two strings entered by the user.

String IX

```c
#include<stdio.h>
#include<string.h>
int main()
{
	char strg1[50], strg2[50];
	printf("Enter first string: ");
	gets(strg1);
	printf("Enter second string: ");
	gets(strg2);
	if(strcmp(strg1, strg2)==0)
	{
		printf("\nYou entered the same string two times");
	}
	else
	{
		printf("\nEntered strings are not same!");
	}
	return 0;
}
```

✓ Our own strcmp function:

String X

```
1  int my_strcmp(char *strg1, char *strg2)
2  {
3        while( ( *strg1 != '\0' && *strg2 != '\0' ) && *strg1 == *strg2 )
4        {
5              strg1++;
6              strg2++;
7        }
8
9        if(*strg1 == *strg2)
10       {
11             return 0; // strings are identical
12       }
13
14       else
15       {
16             return *strg1 - *strg2;
17       }
18 }
```

String XI

The strcpy() Function in C:

```
char* strcpy (char* destination, const char* source);
```

```
char ch_arr[] = "string array";
strcpy("destination string", c_arr); // wrong
```

✓ Our own strcpy function:

```
char *my_strcpy(char *destination, char *source)
{
       char *start = destination;

       while(*source != '\0')
       {
              *destination = *source;
              destination++;
              source++;
       }

       *destination = '\0'; // add '\0' at the end
       return start;
}
```