MODULE 4

TWO DIMENSIONAL ARRAYS

- A two dimensional array has two subscripts/indexes.
- The first subscript refers to the row, and the second, to the column.

For examples,

- int Integer[3][4];
- float matrixNum[20][25];

The first line declares Integer as an integer array with 3 rows and 4 columns.

Second line declares a matrixNum as a floating-point array with 20 rows and 25 columns.

DECLARING TWO DIMENSIONAL ARRAYS

Similar to one-dimensional arrays must be declared before being used. The declaration statement tells the compiler the name of the array, the data type of each element in the array, and the size of each dimension. A two-dimensional array is declared

Syntax : data_type array_name[row_size][column_size]

For example, if we want to store the marks obtained by 3 students in different subjects, then we can declare a two-dimensional array as

Example : int marks[3][5];

A two-dimensional array called marks is declared that has m(3) rows and n(5) columns.

INITIALIZING TWO DIMENSIONAL ARRAYS

Assigning required values to a variables at the time of declaration before processing it is called as Initialization.

A two-dimensional array is initialized in the same way as a one-dimensional array.

For example : int marks[2][3] = {90, 87, 78, 68, 62, 71};

The initialization of a two-dimensional array is done row by row. The above statement can also be written as int marks[2][3]={{90,87,78},{68, 62, 71}};

ACCESSING THE ELEMENTS OF TWO-DIMENSIONAL ARRAYS

The elements of a 2D array are stored in contiguous memory locations. While accessing the elements, remember that the last subscript varies most rapidly whereas the first varies least rapidly.

Look at the programs given below which use two for loops to access the elements of a 2D array.

- Write a program to print the elements of a 2D array.

```
#include <stdio.h>
 #include <conio.h>
int main()
 { int arr[2][2] = {12, 34, 56,32};
 int i, 4;
 for(i<0;i<2;i<++)
{ printf("\n");
 for(j=0;j<2; j++)
printf("xd\t", arr[i][j]); }
return 0; }
```

Output : 12 34
56 32

OPERATIONS ON TWO DIMENSIONAL ARRAYS

Two-dimensional arrays can be used to implement the mathematical concept of matrices. In mathematics, a matrix is a grid of numbers, arranged in rows and columns. Thus, using two-dimensional arrays, we can perform the following operations on an m * n matrix.

- **Transpose** - Transpose of a m x n matrix A is given as a n x m matrix B where,
  $B_{i,j} = A_{j,i}$

- **Sum** -Two matrices that are compatible with each other can be added together thereby storing the result in the third matrix. Two matrices are said to be compatible when they have the same number of rows and columns. Elements of the matrices can be added by writing:
  $C_{i,j} = A_{i,j} + B_{i,j}$

- **Difference** - Two matrices that are compatible with each other can be subtracted thereby storing the result in the third matrix. Two matrices are said to be compatible when they have the same number of rows and columns. Elements of the matrices can be subtracted by writing:
  $C_{i,j} = A_{i,j} - B_{i,j}$

- **Product** - Two matrices can be multiplied with each other if the number of columns in the first matrix is equal to the number of rows in the second matrix.
  Therefore, m x n matrix A can be multiplied with a p x q matrix if n = p.
  Elements of the matrices can be multiplied by writing:
  $C_{i,j} = E\ A_{i,k}\ B_{k,j}$ for k=l to k <n

TWO-DIMENSIONAL ARRAYS TO FUNCTIONS

There are three ways of passing two-dimensional arrays to functions.

1. **First,** we can pass individual elements of the array. This is exactly same as passing elements of a one dimensional array.
2. **Second,** we can pass a single row of the two-dimensional array. This is equivalent to passing the entire one-dimensional array to a function.
3. **Third**, we can pass the entire two-dimensional array to the function.

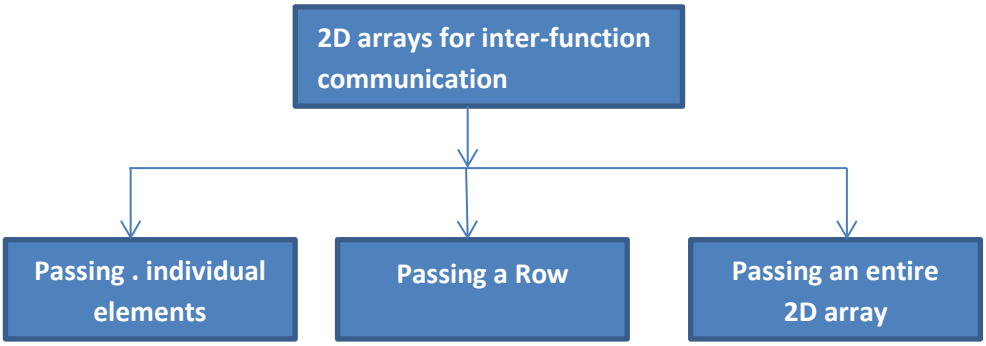


Figure: Two-dimensional arrays for inter-function communication

Passing a Row

A row of a two-dimensional array can be passed by indexing the array name with the row number. When we send a single row of a two-dimensional array, then the called function receives a one-dimensional array. Below figure illustrates how a single row of a two-dimensional array is passed to the called function.

```
Calling function   -> main()
                        { int arr(2}[3]}* ({1, 2, 3), (4, 5, 6));
                      func(arr(1]); )
Called function -> void func(int arr[])
                        { int i;
                      for (i=0;i<3;i++)
                      printf("%d", arr[i] * 10);
```

Figure: Passing a single row of a 2D array

Passing an Entire 2D Array

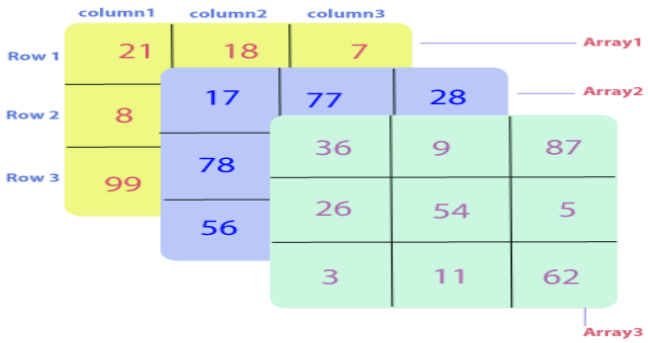
To pass a two-dimensional array to a function, we use the array name as the actual parameter. However, the parameter in the called function must indicate that the array has two dimensions.

Example : A menu-driven program to read and display an m x n matrix, Also to find the sum, transpose, and product of two m * n matrices. (Refer text book)

MULTIDIMENSIONAL ARRAY

- A multidimensional array in simple terms is an array of arrays. Like we have one index in a one-dimensional array, two indices in a two-dimensional array, in the same way we have n indices in an n-dimensional array or multidimensional array.
- Conversely, an n-dimensional array is specified using n indices.
- An n-dimensional m_1 x m_2 x m_3 x….. m_n array is a collection m_1 x m_2 x m_3 x….. m_n elements.
- In a multidimensional array, a particular element is specified by using n subscripts as $A[I_1][I_2][I_3]…..[I_n]$ where,

  $I_1 <= M_1$ $I_2 <= M_2$ $I_3 <= M_3$…….. $I_n <= M_n$
- A multidimensional array can contain as many indices as needed and the requirement of the memory increases with the number of indices used.
- Figure shows a three-dimensional. The array has three pages, three rows, and three columns. A multidimensional array is declared and initialized similar to one- and two-dimensional arrays.



C# Three Dimensional(3D) Array

APPLICATIONS OF ARRAYS

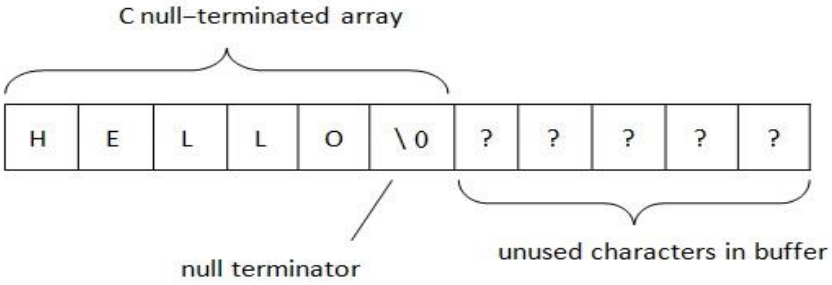
- Arrays are widely used to implement mathematical vectors, matrices, and other kinds of rectangular tables
- Many databases include one-dimensional arrays whose elements are records.
- Arrays are also used to implement other data structures such as strings, stacks, queues, heaps, and hash tables.
- Arrays can be used for sorting elements in ascending or descending order.

INTRODUCTION TO STRINGS

- String is an array of Characters and terminated by NULL character which is denoted by the escape sequence '\0'.
- A null terminated string is the only type of string defined by C.

String is stored in Memory as sequence of character terminated by '\0'

Ex:



DECLARATION OF A STRING

A String is declared like an array of character.

Syntax

data_type string_name[size];

- data type is type of value that array is going to hold that is char string name is name of an array size.

Example: char a[20];

String size 20 means it can store upto 19 characters plus the NULL character.

INITIALIZATION OF STRING

Initialization is the process of assigning values to a variable before doing manipulation. Initialization can be done in various ways -

Initializing location character by character

Char b[9]={'C','O','M','P','U','T','E','R'};

C	O	M	P	U	T	E	R	\0
0	1	2	3	4	5	6	7	8

Compiler allocates 9 memory location ranging from 0 to 8 and these locations are initialized with character and remaining locations are initialized to NULL character automatically.

READING AND WRITING A STRINGS

If we declare a string by writing char str[100];

Then string can be read by using three ways :

1. using scanf function

2. using gets() function

3. using getchar(), getch() or getche() function repeatedly.

Strings can be read using scanf() by writing scanf("%s", str);

For example, if the user enters Hello World, then str will contain only Hello.

This is because the moment a blank space is encountered, the string is terminated by the scanf() function.

i. **Reading a String:** To read a string we use scanf() function with %s string specifier.

Ex: char name[10];

printf("Enter the name\n");
scanf("%s", name);

While reading strings we need not to specify address operator(&) because, character array itself is an address.

Unlike, int(%d), float(%f) and char(%c), %s – string format.

- The next method of reading a string is by using **gets() function.**
- The string can be read by writing gets(str);
- gets() - is a simple function that overcomes the drawbacks of the scanf() function.
- The gets() function takes the starting address of the string which will hold the input.
- The string inputted using gets() is automatically terminated with a null character.

- Strings can also be read by calling the **getchar()** function repeatedly to read a sequence of single characters and simultaneously storing it in a character array as shown below :

```
i=0;
ch = getchar();
 while(ch != '*')

// Get a character

{
Str[i] = ch; // Store the read character in str ;
```

ch = getchar(); // Get another character }

str[i] = '\0'; // terminate str with null character does not require ampersand before the variable name.

Writing/Printing a String:

Strings can be displayed on screen using three ways.

1. using printf() function
2. using puts() function
3. using putchar() function repeatedly

A string can be displayed using **printf()** by writing printf("%s", str);

- We use the conversion character 's' to output a string. We may also use width and precision specifications along with %s.
- The width specifies the minimum output field width. If the string is short, extra space is either left padded or right padded.
- A negative width left pads short string rather than the default right justification.
- The precision specifies the maximum number of characters to be displayed.
- If the string is long, the extra characters are truncated.
  For example, printf("%s.3s", str);

  - The above statement would print only the first three characters in a total field of five characters.
  - Also these three characters are right justified in the allocated width to make the string left justified, we must use a minus sign.
    For example, printf("%-s.3s", str)

We use printf() function with **%s** format specifier to print a string.
It prints the character until NULL character.

- The next method of writing a string is by using **puts()** function. The string can be displayed by writing puts(str);
- puts() is a simple function that overcomes the drawbacks of the printf() function.
- The puts() function writes a line of output on the screen. It terminates the line with a newline character ("\n"). It returns an EOF (~1) if an error occurs and returns a positive number on success.

- Last but not the least, strings can also be written by calling the **putchar()** function repeatedly to print a sequence of single characters.

```
i=0;
while(str[i] != "\0')
 {
  putchar(str[i]); // Print the character on the screen
i++;
}
```

Ex 1: char name[10] = “SUVIKA”;
printf(“The name is %s”, name);

Output: The name is SUVIKA

Ex 2: char name[10] = “SUVIKA”;
printf(“%s”, name);
printf(“%9.3s”, name);
printf(“%-10.3s”, name);

Output:

S	U	V	I	K	A

						S	U	V

S	U	V							

1. Write a program to display a string using printf()

```
#include <stdio.h>
 #include <conio.h>
int main()
{
 char str[] = “Introduction to C”;
 clrscr();
printf("\n |%s|", str);
printf("\n |%20s|", str);
 printf("\n |%-20s|", str);
 printf("\n |%.4s|", str);
 printf("\n |%20.4s|", str);
printf("\n |%-20.4s|", str);
getch();
 return 0;
}
```

Output

```
| Introduction to c|
|        Introduction to c |
| Introduction to C        |
| Intr |
|                        Intr |
|Intr                          |
```

Functions to read and write characters

1. getchar() - Used to read a character from the keyboard; waits for carriage return (enter key), It returns an integer, in which the low-order byte contains the character. getchar() can be used to input any key, including RETURN, TAB, and ESC.
2. getch() - Is un alternative for getchar(), Unlike getchar(), the getch() function waits for a keypress, after which it returns immediately.
3. getche() - Similar to getch(). The only difference is that getche() echoes the character on screen.
4. putchar() - Used to write a character to the screen. It accepts an integer parameter of which only the low-order byte is output to the screen. Returns the character Written, of EOF (~1) if an error occurs.

➢ sprintf() Function

The library function sprintf() is similar to printf().
The only difference is that the formatted output is written to a memory area rather than directly to a standard output (screen).
sprintf() is useful in situations when formatted strings in memory have to be transmitted over a communication channel or to a special device.

The **syntax** of sprintf() can be given by -

int sprintf(char * buffer, const char * format [, argument , ...]);

Here, buffer is the place where string needs to be stored.
The arguments command is an ellipsis so you can put as many types of arguments as you want, Finally, format is the string that contains the text to be printed. The string may contain format tags.

```
#include < stdio.h>
main()
{
char buf[100};
 int num = 10;
sprint(buf, “num+%3d”,num);
}
```

SUPPRESSING INPUT USING A SCANSET.

The scanf() function can be used to read a field without assigning it to any variable. This is done by preceding that field's format code with a *.

For example, consider the example below:

scanf("%d*c%d", &hr, &min);

The time can be read as 9:05. Here the colon would be read but not assigned to anything. Therefore, assignment suppression is particularly useful when part of what is input needs to be suppressed.

Using a Scanset

The ANSI standard added the new scanset feature to the C language.

Scanset is used to define a set of characters which may be read and assigned to the corresponding string.

Scanset is defined by placing the characters inside square brackets prefixed with a %, as shown in the example - %["aeiou"]

When we use the above scanset, scanf() will continue to read characters and put them into the string until it encounters a character that is not specified in the scanset.

For example, consider the code given below.

```
#include <stdio.h>

int main()

{

 char str[10];

 printf("\n Enter string:" );

scanf("%[aeiou]", str );

 printf( "The string is: %s", str);

return 0;

}
```

- The code will stop accepting a character as soon as the user enters a character that is not a vowel.
- However, if the first character in the set is a ^ (caret symbol), then scanf() will accept any character that is not defined by the scanset. For example, if you write scanf("%[*aeiou]", str);
- Then, str will accept characters other than those specified in the scanset, ic, it will accept any non-vowel character.
- However, the caret and the opening bracket can be included in the scanset anywhere.
- They have a predefined meaning only when they are included as the first character of the scanset.
- So if you want to accept a text from the user that contains caret and opening bracket then make sure that they are not the first characters in the scanset.
- This is shown in the following example:

  scanf("%[0123456789.^[ ]()_+-$%&*]",str);

➢ In the given example, str can accept any character enclosed in the opening and closing square brackets (including ^ and [ ).

➢ The user can also specify a range of acceptable characters using a hyphen. This is shown in the statement given: scanf("%[a-z]", str);

- Here, str will accept any character from small a to small z. Always remember that scansets are case sensitive. However, if you want to accept a hyphen then it must either be the first or the last character in the set.

To better understand scanset, try the following code with different inputs

```
#include <stdio.h>
int main()
{
char str[10];
 printf("\n Enter string: " );
scanf("%[A-Z]", str );    // Reads only upper case characters
printf( “The string is : %s", str);
return 0;
}
```

A major difference between scanset and the string conversion codes is that scanset does not skip leading white spaces, If the white space is a part of the scanset, then scanset accepts any white space character, otherwise it terminates if a white space character is entered without being specified in the scanset.

Scanset may also terminate if a field width specification is included and the maximum number of characters that can be read has been reached.

For example, the statement given below will read maximum 10 vowels. So the scanf statement will terminate if 10 characters have been read or if a non-vowel character is entered.

scanf("%10[aeiou)", str);

 # sscanf() Function

The sscanf function accepts a string from which to read input.

It accepts a template string and a series of related arguments.

The sscanf function is similar to scanf function except that the first argument of sscanf specifies a string from which to read, whereas scanf can only read from standard input.

Its syntax is given as - sscanf(const char *str, const char *format,[pl, p2, ...]);

Here sscanf() reads data from str and stores them according to the parameter format into the locations given by the additional arguments.

Consider the example given below

sscanf(str, "Xd", &num);

Here, sscanf takes three arguments.

- The first is str that contains data to be converted.
- The second is a string containing a format specifier that determines how the string is converted.
- Finally, the third is a memory location to place the result of the conversion. When the sscanf function completes successfully, it returns the number of items successfully read.

Similar to scanf(), sscanf() terminates as soon as it encounters a space, i.c., it continues to read ll it comes across a blank space