DESIGNING STRUCTURED PROGRAMS

Whenever we are solving large programs first, we must understand the problem as a whole, then we must break it in to simpler, understandable parts. We call each of these parts of a program a module and the process of sub dividing a problem into manageable parts top-down design.

- The principles of top-don design and structured programming dictate that a program should be divided into a main module and its related modules.
- The division of modules proceeds until the module consists only of elementary process that is intrinsically understood and cannot be further subdivided. This process is known as factoring.
- Top-down design is usually done using a visual representation of the modules known as a structured chart.

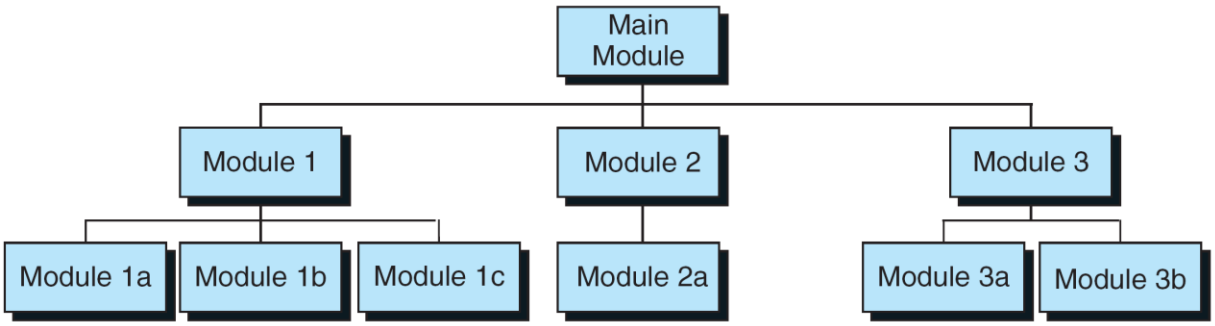


Figure: Structure Chart

FUNCTIONS IN C

A function is a self-contained block of code that carries out some specific and well-defined task.

C functions are classified into two categories

1. Library Functions
2. User Defined Functions

Library Functions

These are the built in functions available in standard library of C.The standard C library is collection various types of functions which perform some standard and predefined tasks.

Example: abs (a) function gives the absolute value of a, available in <math.h> header file

pow (x, y) function computes x power y. available in <math.h> header file

printf ()/scanf () performs I/O functions. Etc..,

User Defined Functions

These functions are written by the programmer to perform some specific tasks.

Example: main (), sum (), fact () etc.

The Major distinction between these two categories is that library functions are not required to be written by us whereas a user defined function has to be developed by the user at the time of writing a program.

USER-DEFINED FUNCTIONS

The basic philosophy of function is divide and conquer by which a complicated tasks are successively divided into simpler and more manageable tasks which can be easily handled. A program can be divided into smaller subprograms that can be developed and tested successfully.

A function is a complete and independent program which is used (or invoked) by the main program or other subprograms. A subprogram receives values called arguments from a calling program, performs calculations and returns the results to the calling program.

Def : A function is a self contained block that carries out a specific well defined task.

Advantages

1. Reusability i.e. a function may be used by many other programs.
2. The length of the source program can be reduced.
3. It is easy to locate and isolate a faulty function.
4. It facilitates top-down modular programming.

Communications Among functions

Functions communicate among themselves using arguments and return value.

Farm of 'C' function

return-datatype function name (argument list)

```
{
        local variable declarations;
        executable  statements(s);
        return (expression);
}
```

eg: void mul (int x, int y)

```
{

        int     p;
        p=x*y;

        printf("product = %d",p);

}
```

Return values and their types

A function may (or) may not send back any value to the calling function if it does, it is done through the return statement.
A called function can only return one value per call the return types are void, int, float, char and double.

If a function is not returning any value then its return type is 'void'

Function name

A function must follow the same rules of formation as other variables name in 'C'

A function name must not duplicate library routine names (or) predefined function names.

Argument list

The argument list contains valid variable names separated by commas

The argument variables receive values form the calling function, thus providing a means for data communication from the calling function to the called function.

Calling a function

A function can be called by simply using the function name in a statement

Function definition

When the compiler encounters a function call, the control is transferred to the function definition.

All the statements, in the called function, are together called as function definition

Function header

The first line in the function definition is called function header.

Actual parameter

All the variables inside the function call are called actual parameters.

Formal parameters

All the variable inside the function header are called formal parameters

Program

```
#include<stdio.h>

#include<conio.h>

main ( )

{

        int mul (int, int);  ——→  function prototype

        int a,b,c;

        clrscr( );

        printf ("enter 2 numbers");

        scanf("%d %d, &a, &b);

        c = mul (a,b);  ——————→ function call

        printf("product =%d",c);   Actual parameters

        getch ( );

}

        int mul (int a, int b)          Formal parameters

        {                               function header

                int c;

                c = a *b;  ——→ Function definition

                return c;

        }
```

Output

Enter 2 numbers: 10 20

Product = 200

Categories of functions:

Depending on whether arguments are present (or) not and whether a value is returned (or) not, functions are categorized into:

1) functions without arguments and without return values
2) functions without arguments and with return values
3) Functions with arguments and without return values
4) Functions with arguments and with return values.

1) functions without arguments and without return values

Calling function	Analysis	Called function

main ( )
{
--- 	
--- 	
fun ( );
}
	No arguments are passed
	
	No values are sent back
		fun ( )
		{
		---- 	
		---- 	
		}

eg:

```
main ( )
{
        void sum ( );
        clrscr ( );
        sum ( );
        getch ( );
}
void sum ( )
{
        int a,b,c;
        printf("enter 2 numbers");
        scanf ("%d%d", &a, &b);
        c = a+b;
        printf("sum = %d",c);
}
```

Output

```
Enter 2 numbers
10  20
Sum=30
```

2) functions without arguments and with return values

Calling function	Analysis	Called function

main ( )
{
int c;
--- 	
c= fun ( );
----- 	
----- 	
}
	No arguments are passed
	
	values are sent back ←
		fun ( )
		{
		---- 	
		---- 	
		return c;
		}

eg:

```
main ( )
{
        int sum ( );
        int c;
        clrscr ( );
        c= sum ( );
        printf("sum = %d",c);
        getch ( );
}
int sum ( )
{
        int a,b,c;
        printf("enter 2 numbers");
        scanf ("%d%d", &a, &b);
        c = a+b;
        return c;
}
```

Output

```
enter 2 numbers
  10   20
Sum =  30
```

THE RETURN STATEMENT

- The return statement is the mechanism for returning a value from the called function to its caller.

- The general form of the return statement is

  return expression;

- The calling function is free to ignore the returned value. Furthermore, there need not be expression after the return.

- In any case if a function fails to return a value, its value is certain to be garbage.

- The return statement has two important uses
  1. It causes an immediate exit of the control from the function. That is, it causes program execution to return to the calling function.

  2. It returns the value present in the expression.

     example: return(x+y);
     return (6*8);
     return (3);
     return;

3) Functions with arguments and without return values

Calling function	Analysis	Called function

main ( )
{
--- 	
--- 	
fun (a,b);
--- 	
--- 	
}
	Arguments are passed
	
	No values are sent back ←
		fun (int a, int b)
		{
		---- 	
		---- 	
		}

eg:

```
main ( )
{
        void sum (int, int );
        int a,b;
        clrscr ( );
        printf("enter 2 numbers");
        scanf("%d%d", &a,&b);
        sum (a,b);
        getch ( );
}
void sum ( int a, int b)
{
        int c;
        c= a+b;
        printf ("sum=%d", c);
}
```

Output

```
enter 2 numbers
10 20
sum = 30
```

4) Functions with arguments and with return values.

Calling function	Analysis	Called function

main ( )
{
int c;
--- 	
--- 	
c= fun (a,b);
--- 	
--- 	
}
	Arguments are passed
	
	value are sent back ←
		fun (int a, int b)
		{
		---- 	
		---- 	
		return c;
		}

eg:

```
main ( )
{
        int sum ( );
        int a,b,c;
        clrscr ( );
        printf("enter 2 numbers");
        scanf("%d%d", &a,&b);
        c= sum (a,b);
        printf ("sum=%d", c);
        getch ( );
}
int sum ( int a, int b )
{
        int c;
        c= a+b;
        return c;
}
```

Output

```
enter 2 numbers
10   20
Sum =  30
```

Scope

- "scope" of a variable determines the part of the program where it is visible

2 types

1. local scope
2. global scope

1. local scope

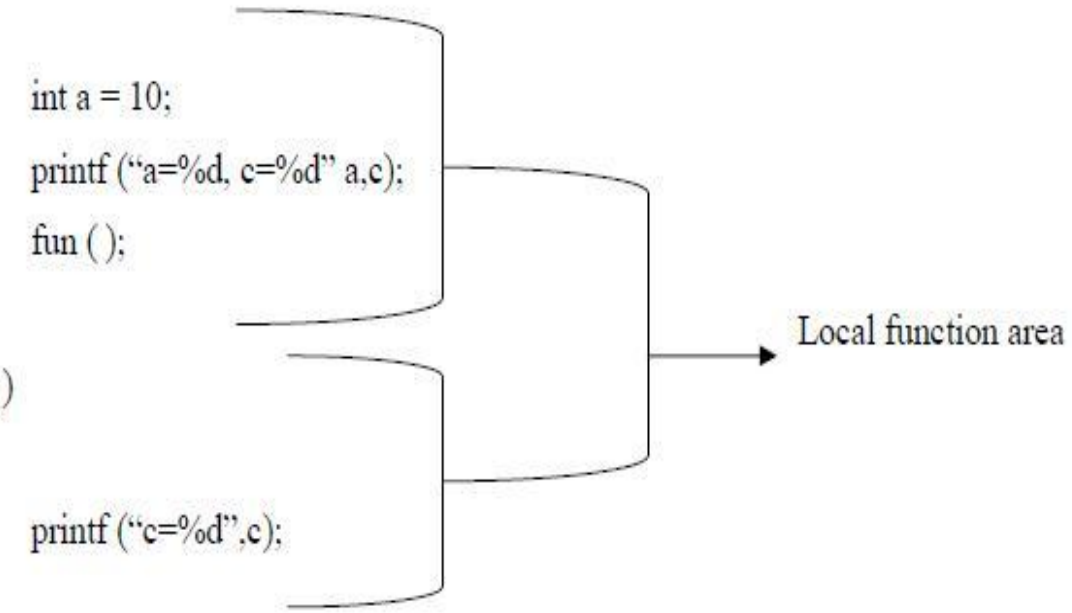
- Local scope specifies that variables defined within the block are visible only in that block and invisible outside the block.

2. global scope

- Global scope specifies that variables defined outside the block are visible upto end of the program.

eg:

```
int c= 30        /* global area */
main ( )
{
        int a = 10;
        printf ("a=%d, c=%d" a,c);
        fun ( );
}
fun ( )
{
        printf ("c=%d",c);
}
```



output

a =10, c = 30

c = 30

ARRAYS IN C

INTRODUCTION

- Often we need many different variables that are of the same type and play a similar role in the program
- For example, suppose we have a temperature reading for each day of the year and need to manipulate these values several times within a program.
- With simple variables,
  - We would need to declare 365 variables.
  - We would not be able to loop over these variables.
- An array is a numbered collection of variables of the same type.
- An array is a homogeneous collection of data.

- The variables in an array share the same name and are distinguished from one another by their numbers or subscripts.
- We can loop through the variables of an array by using a variable for the subscript.
- The subscripts are always 0,1,…, size-1

ARRAY CHARACTERISTICS

- An array represents a group of related data.
- An array has two distinct characteristics:

An array is ordered: data is grouped sequentially. In other words, here is element 0, element 1, etc.

An array is homogenous: every value within the array must share the same data type. In other words, an int array can only hold ints, not doubles.

HOW TO CREATE AN ARRAY

- Before we create an array, we need to decide on two properties:

Element type: what kind of data will your array hold? int, double, char? Remember, we can only choose one type.

Array size: how many elements will your array contain?

When we initially write our program, we must pick an array size. An array will stay this size throughout the execution of the program. In other words, we can change the size of an array at compile time, but cannot change it at run-time.

USING ARRAYS IN C

In C, the arrays can be classified based on how the data items are arranged for human understanding

Arrays are broadly classified into three categories,

1. One dimensional arrays
2. Two dimensional arrays
3. Multi dimensional arrays

ONE DIMENSIONAL ARRAY

One dimensional array is a linear list consisting of related and similar data items. In memory all the data items are stored in contiguous memory locations one after the other.

DECLARING ONE DIMENSIONAL ARRAYS

- To declare regular variables we just specify a data type and a unique name:

int number;

- To declare an array, we just add an array size.
- For example:

int temp[5];

Creates an array of 5 integer elements.

- For example:

double stockprice[31];

creates an array of 31 doubles.

Syntax for Declaring one dimensional Arrays

elementType arrayName [size];

Where :

elementType: specifies data type of each element in the array.

arrayName: specifies name of the variable you are declaring.

size: specifies number of elements allocated for this array.

INITIALIZING ONE DIMENSIONAL ARRAYS

- We have four options for initializing one dimensional arrays.
- But, first, REMEMBER THIS: Just like regular variables, arrays that have not been initialized may contain "garbage values."
- But, now, the stakes are even higher. If you declare an array with 31 elements, and forget to initialize it, you end up with 31 garbage values!

Option 1 Initializing all memory locations

If you know all the data at compile time, you can specify all your data within brackets:

int temp [5] = {75, 79, 82, 70, 68};

during compilation, 5 contiguous memory locations are reserved by the compiler for the variable temp and all these locations are initialized as shown in Fig.

If the size of integer is 2 bytes, 10 bytes will be allocated for the variable temp.

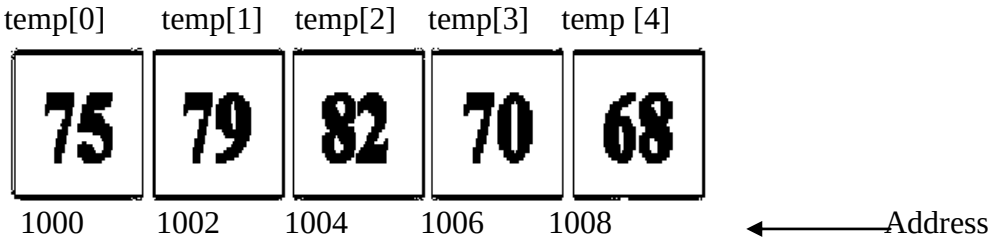


Figure: Storage Representation of array

Option 2 initialization without size

If we omit the size of your array, but specify an initial set of data, the compiler will automatically determine the size of your array. This way is referred as initialization without size.

int temp [] = {75, 79, 82, 70, 68};

In this declaration, even though we have not specified exact number of elements to be used in array temp, the array size will be set of the total number of initial values specified. Here, the compiler creates an array of 5 elements. The array temp is initialized as shown in Figure .

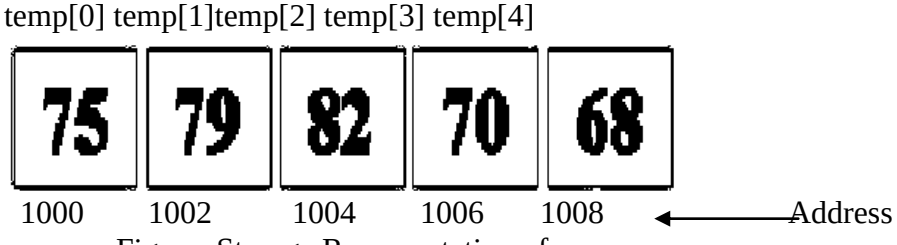


Figure: Storage Representation of array

Option 3 Partial Array Initialization

If the number of values to be initialized is less than the size of the array, then the elements are initialized in the order from 0^{th} location. The remaining locations will be initialized to zero automatically.

int temp [5] = {75, 79, 82};

Even though compiler allocates 5 memory locations, using this declaration statement, the compiler initializes first three locations with 75,70 and 82,the next set of memory locations are automatically initialized to 0's by the compiler as shown in figure

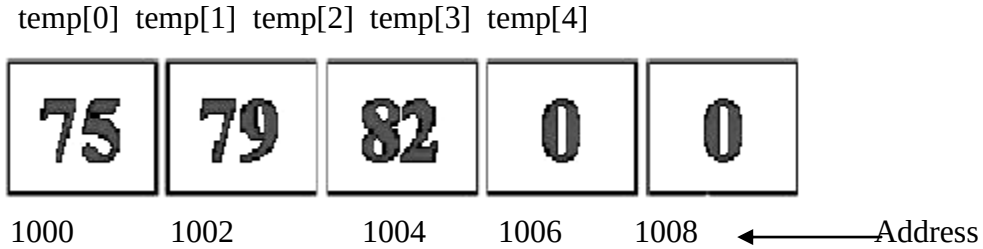


Figure: Storage Representation of array

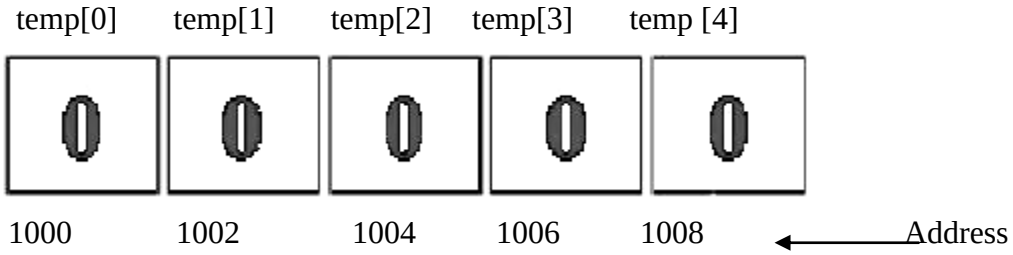
Option 4

If you do not know any data ahead of time, but you want to initialize everything to 0, just use 0 within { }.

For example:

int temp [5] = {0};

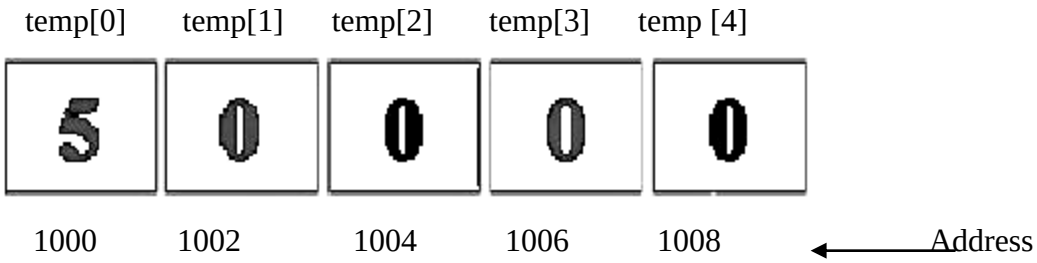
This will initialize every element within the array to 0 as shown in below figure.



If at least one element is assigned then only one element is assigned with the value and all the remaining elements of the array will be zero.

For example:

int temp [5] = {5};

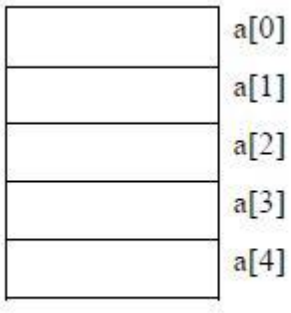


The first value is supplied in the first element memory location, remaining all elements are placed with zero.

Eg: To represent a set of 5 numbers by an array, it can be declared as follows

int a[5];

Then computer reserves 5 storage locations each of 2 bytes.



First element is identified by subscript 'zero' i.e., a[0] represents first element of the array.

If there are 'n' elements in array then subscripts range from 0 to n-1

Initialization

To store values into an array it can be done as follows.

a[0] = 10;	10	a[0]
a[1] = 20;	20	a[1]
a[2] = 30;	30	a[2]
a[3] = 40;	40	a[3]
a[4] = 50;	50	a[4]

An array can also be initialized at the time of declaration as follows

int a[5] = { 10,20,30,40,50};

Program for compile time initialization and sequential access using for loop

```
main ( )
{
        int a[5] = {10,20,30,40,50};
        int i;
        clrscr ( );
        printf ("elements of the array are");
        for ( i=0; i<5; i++)
                printf ("%d, a[i]);
        getch ( );
}
```

storing

accessing

a[0]	10
a[1]	20
a[2]	30
a[3]	40
a[4]	50

Output: Elements of the array are

10 20 30 40 50

Program for runtime initialization and sequential access using for loop

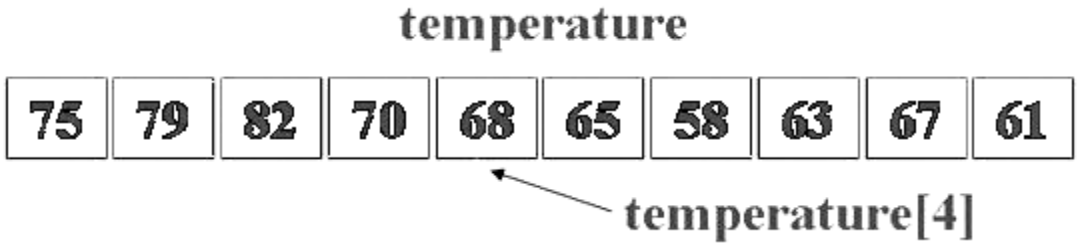
```
main ( )
{
int a[5],i;
clrscr ( );
printf (“enter 5 elements”);
for ( i=0; i<5; i++)
scanf(”%d”, &a[i]); //Storing / assigning values to element of an array
printf(“elements of the array are”);
for (i=0; i<5; i++)
printf(”%d ”, a[i]);      //Accessing the elements of the aray
getch ( );
}
```

output

enter 5 elements 10 20 30 40 50
elements of the array are : 10 20 30 40 50

REFERENCING/ACCESSING ONE DIMENSIONAL ARRAY ELEMENTS

- Suppose we have an array, temperature, for the temperature readings for the year.
- The subscripts would be 0,1,…,364.
- To refer to the reading for a given day, we use the name of the array with the subscript in brackets: temperature [4] for the fifth day.



- To assign the value 89 for the 150th day:

  temperature [149] = 89;

- We can loop through the elements of an array by varying the subscript. To set all of the values to 0, say

```
for(i=0;i<365;i++)
    temperature[i] = 0;
```

- We cannot use assignment statement directly with arrays. if a[] , b[] are two arrays then the assignment a=b is not valid .

```
// C program to read an array elements and find the sum of the elements.
#include <stdio.h>
void main()
{
        int a[10];
        int i, SIZE, total=0;
        printf("\n Enter the size of the array : ");
        scanf("%d",&size);
        for (i = 0; i < SIZE ; i++)
                scanf("%d",&a[i]);
        for (i = 0; i < SIZE ; i++)
                total += a[i];
        printf("Total of array element values is %d\n", total);
        getch();
}
```

- C does not provide array bounds checking.
- Hence, if we have

```
double stockPrice[5];
printf ("%d", stockPrice[10]);
```

- This will compile, but you have overstepped the bounds of your array. You may therefore be accessing another variable in your program or another application.

TWO DIMENSIONAL ARRAYS

- An array of arrays is called a two-dimensional array and can be regarded as a table with a number of rows and columns:
-

'J'	'o'	'h'	'n'
'M'	'a'	'r'	'v'
'I'	'v'	'a'	'n'

is a 3 × 4 array:
3 rows, 4 columns

'J'	'o'	'h'	'n'
'M'	'a'	'r'	'v'
'I'	'v'	'a'	'n'

-
-
- This is an array of size 3 **names [3]** whose elements are arrays of size 4 **[4]** whose elements are characters **char**

DECLARATION OF TWO DIMENSIONAL ARRAYS

elementType arrayName [rows][cols];

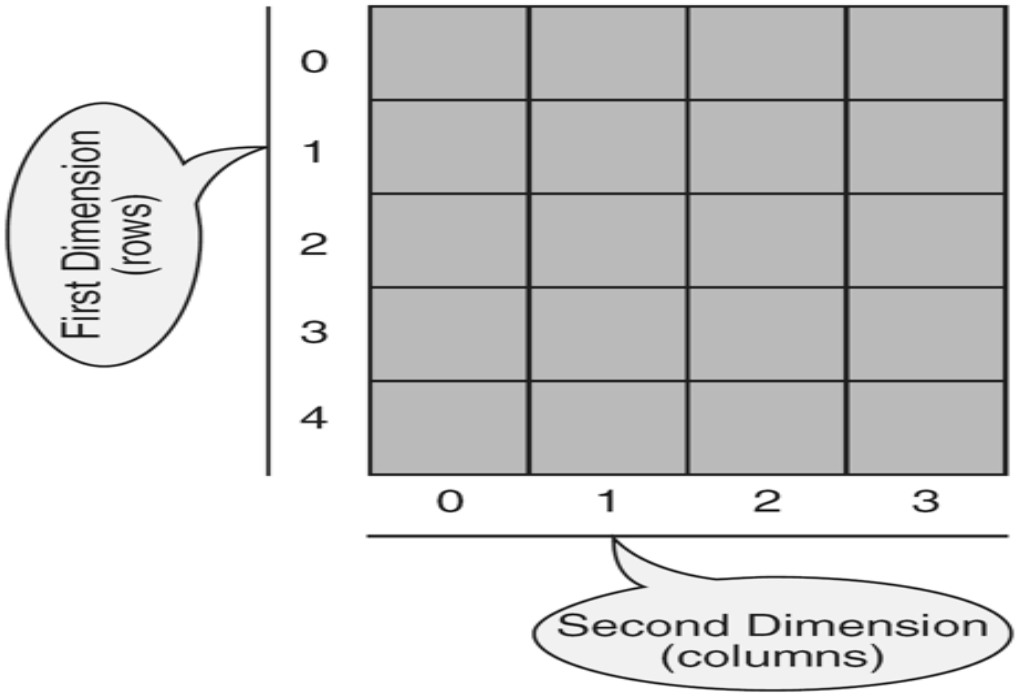


Figure: Two-dimensional Array representation

The above figure shows the representation of elements in the two-dimensional array

Example,

char names[3][4];

in the above declaration

char(elementType) → specifies type of element in each slot
names(arrayName) → specifies name of the array
[3](rows) → specifies number of rows
[4](cols) → specifies number of columns

INITIALIZATION OF TWO DIMENSIONAL ARRAYS

- An array may be initialized at the time of declaration:

```
char names [3][4] = {
                        {'J', 'o', 'h', 'n'},
                        {'M', 'a', 'r', 'y'},
                        {'I', 'v', 'a', 'n'}
                    };
```

- An integer array may be initialized to all zeros as follows

```
int nums [3][4] = {0};
```

- In the declaration of two dimensional array the column size should be specifed, to that it can arrange the elements in the form of rows and columns.
- Two-dimensional arrays in C are stored in "row-major format": the array is laid out contiguously, one row at a time:
- To access an element of a 2D array, you need to specify both the row and the column:

```
nums[0][0] = 16;
printf ("%d", nums[1][2]);
```

Initialization :

Program for compile time initialization and sequential access using nested for loop

```
main ( )
{
    int a[3][3] = {10,20,30,40,50,60,70,80,90};
     int i,j;
    clrscr ( );
    printf (“elements of the array are”);
    for ( i=0; i<3; i++)
    {
        for (j=0;j<3; j++)
        {
            printf(“%d \t”, a[i] [j]);
        }
        printf(“\n”);
    }
    getch ( );
}
```

a[0] [0]	a[0] [1]	a[0] [2]
10	20	30
a[1 [0]	a[1] [1]	a[1] [2]
40	50	60
a[2] [0]	a[2] [1]	a[2] [2]
70	80	90

output

```
elements of the array are:
    10    20    30
    40    50    60
    70    80    90
```

Program for runtime initialization and sequential access using nested for loop

```
main ( )
{
        int a[3][3] ,i,j;
        clrscr ( );
        printf (“enter elements of array”);
        for ( i=0; i<3; i++)
        {
                for (j=0;j<3; j++)
                {
                        scanf(“%d ”, &a[i] [j]);
                }
        }
        printf(“elements of the array are”);
        for ( i=0; i<3; i++)
        {
        for (j=0;j<3; j++)


        {
        printf(“%d\t ”, a[i] [j]);
        }
        printf(“\n”)
        }
        getch( );
}
```

output

```
Enter elements of array : 1  2        3       4       5       6       7       8       9
Elements of the array are
        1       2       3
        4       5       6
        7       8       9
```

FUNCTIONS WITH ARRAYS

Like the values of variable, it is also possible to pass values of an array to a function. To pass an array to a called function, it is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments.

for example, the call

```
findMax(a,n);
```

will pass all the elements contained in the array a of size n.The called function expecting this must be appropriately defined.
The findMax function header looks like:

```
int findMax(int x[],int size);
```

The pair of brackets informs the compiler that the argument x is an array of numbers. It is not necessary to specify the size of the array here.

The function prototype takes of the form

```
int findMax (int [], int );
int findMax (int a [], int );
```

Note: The same thing is applied for passing a multidimensional array to a function. Here we need to specify the column size, for example to read two-dimensional array elements using functions it takes of the form

```
int readElements (int [] [5], int ,int );
```

Arrays and functions

There are 2 ways of passing arrays as arguments to functions. They are

1) sending entire array as argument to function
2) sending individual elements as argument to function.

1. sending entire array as argument to function

To send entire array as argument, just send the array name in the function call. To receive the entire array, an array must be declared in the function header.

Eg:

```
main ( )
{
```

```
        void display (int a[5]);
        int a[5], i;
        clrscr( );
        printf (“enter 5 elements”);
        for (i=0; i<5; i++)
                scanf(“%d”, &a[i]);
        display (a);
        getch( );
}
void display (int a[5])
{
        int i;
        printf (“elements of the array are”);
        for (i=0; i<5; i++)
                printf(‘%d ”, a[i]);
}
```

Staring entire array ‘a’ using array name

Receiving entire array

Output

Enter 5 elements

10 20 30 40 50

Elements of the array are

10 20 30 40 50

2. sending individual elements as argument to function.

If individual elements are to be passed as arguments then array elements along with their subscripts must be given in function call

To receive the elements, simple variables are used in function definition

```
main ( )
{
        void display (int, int);
        int a[5], i;
        clrscr( );
        printf (“enter 5 elements”);
        for (i=0; i<5; i++)
                scanf(“%d”, &a[i]);
```

```
        display (a [0], a[4]);
        getch( );
}
void display (int a, int b)
{
print f ("first element = %d",a);
 printf ("last element = %d",b);
}
```

Output

Enter 5 elements

10 20 30 40 50

First element = 10

Last element = 50

MULTI DIMENSIONAL ARRAYS

- C allows three or more dimensions. The exact limit is determined by the compile.
- The general form of multidimensional array is

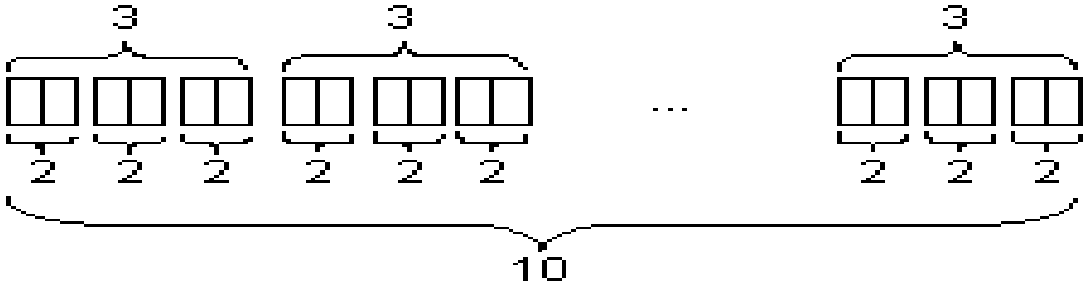
```
type arrayName[s1][s2][s3]….[sm];
```

Where si is the size of the ith dimension.

- Array declarations read right-to-left
- For example

```
int a[10][3][2];
```

- it is represented as "an array of ten arrays of three arrays of two ints"
- In memory the elements are stored as shown in below figure



Syntax:

datatype arrayname [size1] [size2] ----- [sizen];

eg: for 3 – dimensional array:

int a[3] [3] [3]

No of elements = 3*3*3 = 27 elements

Prorgam

```
main ( )
{
    int a[2][2] [2] = {1,2,3,4,5,6,7,8};
    int i,j,k;
    clrscr ( );
    printf ("elements of the array are");
    for ( i=0; i<2; i++)
    {
        for (j=0;j<2; j++)
        {
            for (k=0;k<2; k++)
            {
                printf("%d ", a[i] [j] [k]);
            }
        }
    }
    getch( );
}
```

Output : Elements of the array are :

1 2 3 4 5 6 7 8