Unit-2

FUNCTIONS AND ARRAYS

Function is a self-contained block of statements that can be executed repeatedly whenever we need it.

Benefits of using the function in C

- The function provides modularity.
- The function provides reusable code.
- In large programs, debugging and editing tasks is easy with the use of functions.
- The program can be modularized into smaller parts.
- Separate function independently can be developed according to the needs.

There are two types of functions in C

- Built-in(Library) Functions
  - The system provided these functions and stored in the library. Therefore it is also called *Library Functions*.
    e.g. scanf(), printf(), strcpy, strlwr, strcmp, strlen, strcat etc.
  - To use these functions, you just need to include the appropriate C header files.
- User Defined Functions These functions are defined by the user at the time of writing the program.

Parts of Function

1. Function Prototype (function declaration)
2. Function Definition
3. Function Call

1. Function Prototype

Syntax:

```
dataType functionName (Parameter List)
```

Example:

```
int addition();
```

2. Function Definition

Syntax:

```
returnType functionName(Function arguments){

  //body of the function

}
```

Example:

```
int addition()

{
```

```
}
```

3. Calling a function in C

Program to illustrate the Addition of Two Numbers using User Defined Function

Example:

```
#include<stdio.h>

/* function declaration */int addition();

int main()
{
    /* local variable definition */    int answer;

    /* calling a function to get addition value */    answer = addition();

    printf("The addition of the two numbers is: %d\n",answer);
    return 0;
}

/* function returning the addition of two numbers */int addition()
{
    /* local variable definition */    int num1 = 10, num2 = 5;
    return num1+num2;
}
```

Program Output:

The addition of the two numbers is: 15

Function Arguments

While calling a function, the arguments can be passed to a function in two ways, Call by value and call by reference.

Type	Description
Call by Value	The actual parameter is passed to a function. New memory area created for the passed parameters, can be used only within the function.

	The actual parameters cannot be modified here.
Call by Reference	Instead of copying variable; an address is passed to function as parameters. Address operator(&) is used in the parameter of the called function. Changes in function reflect the change of the original variables.

1. Call by Value

2. Call by Reference

Call by Value

Example:

```
#include<stdio.h>


/* function declaration */int addition(int num1, int num2);


int main()

{

    /* local variable definition */    int answer;

    int num1 = 10;

    int num2 = 5;


    /* calling a function to get addition value */    answer =
addition(num1,num2);


    printf("The addition of two numbers is: %d\n",answer);

    return 0;

}


/* function returning the addition of two numbers */int addition(int a,int
b)

{

    return a + b;

}
```

Program Output:

```
The addition of two numbers is: 15
```

Call By Reference:

Example

```
#include<stdio.h>


/* function declaration */int addition(int *num1, int *num2);


int main()

{

    /* local variable definition */    int answer;

    int num1 = 10;

    int num2 = 5;


    /* calling a function to get addition value */    answer =
addition(&num1,&num2);


    printf("The addition of two numbers is: %d\n",answer);

    return 0;

}


/* function returning the addition of two numbers */int addition(int *a,int
*b)

{

    return *a + *b;

}
```

Program Output:

```
The addition of two numbers is: 15
```

Array

The array is a data structure in C programming, which can store a fixed-size sequential collection of elements of the same data type.For example, if you want to store ten numbers, it is easier to define an array of 10 lengths, instead of defining ten variables.

In the C programming language, an array can be *One-Dimensional*, *Two-Dimensional,* and *Multidimensional*.

Table of Contents

- . Define an Array in C
- . Initialize an Array in C

- A Pictorial Representation of the Array:
- Accessing Array Elements in C

Define an Array in C

Syntax:

```
type arrayName [ size ];
```

This is called a one-dimensional array. An array type can be any valid C data types, and array size must be an integer constant greater than zero.

Example:

```
double amount[5];
```

Initialize an array

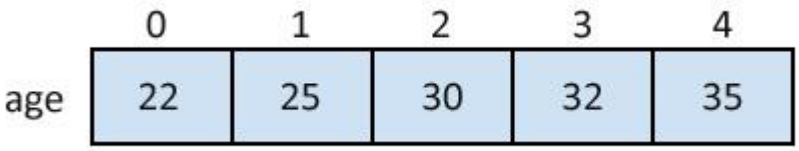
Arrays can be initialized at declaration time:

```
int age[5]={22,25,30,32,35};
```

Initializing each element separately in a loop:

```
int myArray[5];

int n = 0;


// Initializing elements of array seperately

for(n=0;n<sizeof(myArray)/sizeof(myArray[0]);n++)

{

  myArray[n] = n;

}
```

A Pictorial Representation of the Array:



Accessing Array Elements in C

Example:

```
int myArray[5];

int n = 0;


// Initializing elements of array seperately
```

```
for(n=0;n<sizeof(myArray)/sizeof(myArray[0]);n++)
{
  myArray[n] = n;
}

int a = myArray[3]; // Assigning 3rd element of array value to integer 'a'.
```

In C programming, the one-dimensional array of characters are called strings, which is terminated by a null character '\0'.

There are two ways to declare a string in C programming:
Example:

Through an array of characters.

```
char name[6];
```

Through pointers.

```
char *name;
```

Strings Initialization in C

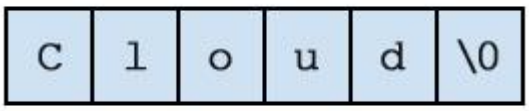
Example:

```
char name[6] = {'C', 'l', 'o', 'u', 'd', '
char name[6] = {'C', 'l', 'o', 'u', 'd', '\0'};
'};
```

or

```
char name[] = "Cloud";
```

Memory Representation of Above Defined String in C



Example:

```
#include<stdio.h>

int main ()
{
```

```
   char name[6] = {'C', 'l', 'o', 'u', 'd', '
#include<stdio.h>


int main ()
{
   char name[6] = {'C', 'l', 'o', 'u', 'd', '\0'};


   printf("Tutorials%s\n", name );


   return 0;
}
'};
   printf("Tutorials%s\n", name );
   return 0;
}
```

Program Output:

```
TutorialsCloud
```

Two dimensional (2D) arrays in C programming with example

An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns. Before we discuss more about two Dimensional array lets have a look at the following C program.

Simple Two dimensional(2D) Array Example

This program demonstrates how to store the elements entered by user in a 2d array and how to display the elements of a two dimensional array.

```
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int disp[2][3];
   /*Counter variables for the loop*/
   int i, j;
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("Enter value for disp[%d][%d]:", i, j);
         scanf("%d", &disp[i][j]);
      }
   }
   //Displaying array elements
   printf("Two Dimensional array elements:\n");
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("%d ", disp[i][j]);
         if(j==2){
            printf("\n");
         }
      }
   }
   return 0;
}
```

Output:

```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

Initialization of 2D Array

There are two ways to initialize a two Dimensional arrays during declaration.

```
int disp[2][4] = {
    {10, 11, 12, 13},
    {14, 15, 16, 17}
};
```

OR

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Things that you must consider while initializing a 2D array

We already know, when we initialize a normal array (or you can say one dimensional array) during declaration, we need not to specify the size of it. However that's not the case with 2D array, you must always specify the second dimension even if you are specifying elements during the declaration. Let's understand this with the help of few examples –

```
/* Valid declaration*/
int abc[2][2] = {1, 2, 3 ,4 }
/* Valid declaration*/
int abc[][2] = {1, 2, 3 ,4 }
/* Invalid declaration – you must specify second dimension*/
int abc[][] = {1, 2, 3 ,4 }
/* Invalid because of the same reason  mentioned above*/
int abc[2][] = {1, 2, 3 ,4 }
```

What is the Difference Between 1D and 2D Array

The **main difference** between 1D and 2D array is that **the 1D array represents multiple data items as a list while 2D array represents multiple data items as a table consisting of rows and columns.**

A variable is a memory location to store data of a specific type. Sometimes, it is necessary to store a set of items of the same data type. An array allows storing multiple items of the same data type. The elements in the array are in subsequent memory locations. There are two types of arrays as one dimensional (1D) array and two dimensional (multi-dimensional) arrays.

Key Areas Covered

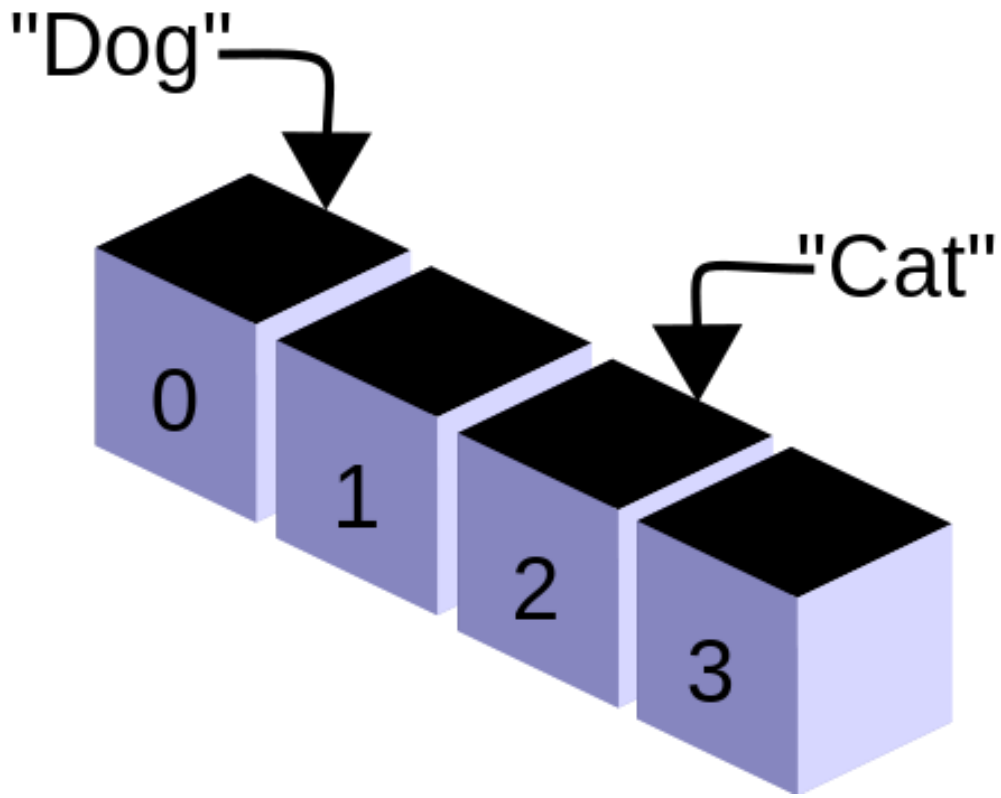
1. **What is 1D Array**
   – *Definition, Functionality*
2. **What is 2D Array**
   – *Definition, Functionality*
3. **What is the Difference Between 1D and 2D Array**
   – *Comparison of Key Differences*

Key Terms

1D Array, 2D Array, Array, Multi-Dimensional Array, Single Dimensional Array

What is 1D Array

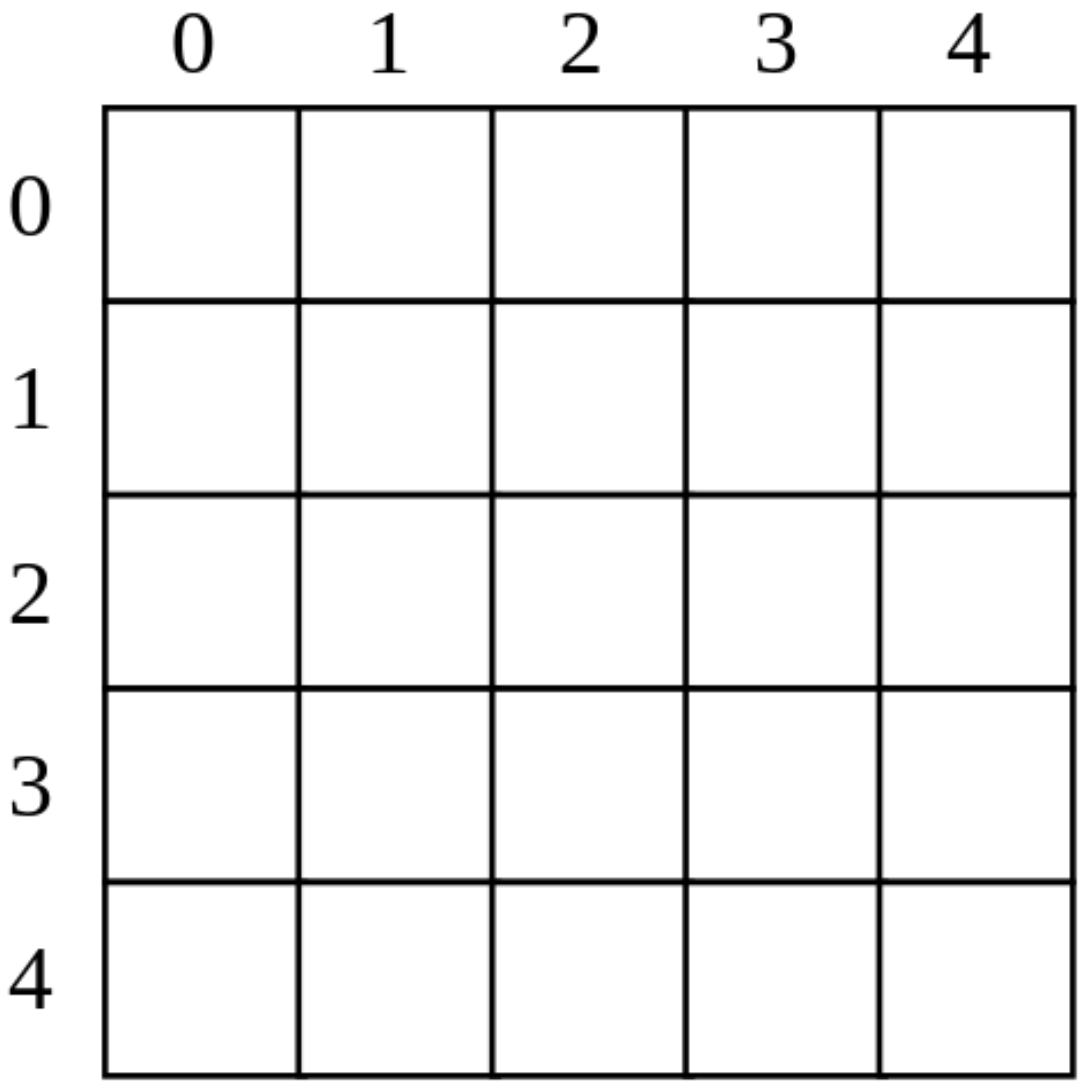
1D array or ***single dimensional array*** stores a list of variables of the same data type. It is possible to access each variable using the index.



int[] numbers; declares an array called numbers..

What is 2D Array

2D array or **multi-dimensional array** stores data in a format consisting of rows and columns.



For example, int[2][3] numbers; declares a 2D arrays of 2 rows and 3 columns

One dimensional Array in C

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.

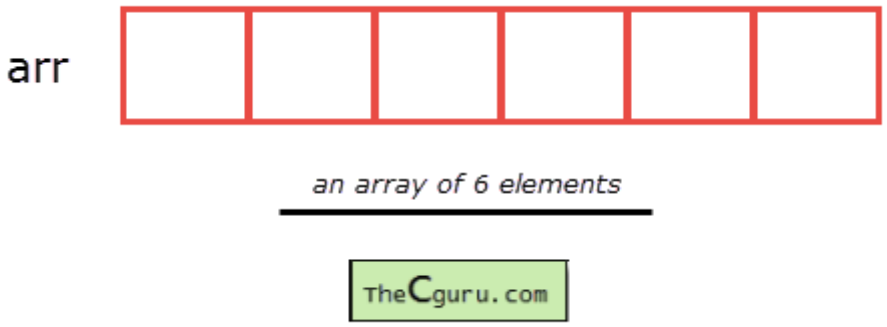
To store roll no. of `100` students, we have to declare an array of size `100` i.e `roll_no[100]`. Here size of the array is `100` , so it is capable of storing `100` values. In C, index or subscript starts from `0`, so `roll_no[0]` is the first element, `roll_no[1]` is the second element and so on. 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 can be single or multidimensional. The number of subscript or index determines the dimensions of the array. An array of one dimension is known as a one-dimensional array or 1-D array, while an array of two dimensions is known as a two-dimensional array or 2-D array.

Let's start with a one-dimensional array.

One-dimensional array

Conceptually you can think of a one-dimensional array as a row, where elements are stored one after another.



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:

```
1  int num[100];
2  float temp[20];
3  char ch[50];
```

`num` is an array of type `int`, which can only store `100` elements of type `int`.
`temp` is an array of type `float`, which can only store `20` elements of type `float`.
`ch` is an array of type `char`, which can only store `50` elements of type `char`.

Note: When an array is declared it contains garbage values.
The individual elements in the array:

```
1  num[0], num[1], num[2], ....., num[99]
```

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

We can also use variables and symbolic constants to specify the size of the array.

```
#define SIZE 10

int main()
{
    int size = 10;

    int my_arr1[SIZE]; // ok
    int my_arr2[size]; // not allowed until C99
    // ...
}
```

Note: Until C99 standard, we were not allowed to use variables to specify the size of the array. If you are using a compiler which supports C99 standard, the above code would compile successfully. However, If you're using an older version of C compiler like Turbo C++, then you will get an error.

The use of symbolic constants makes the program maintainable, because later if you want to change the size of the array you need to modify it at once place only i.e in the `#define` directive.

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*.

```
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
```

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. 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

The following program uses for loop to take input and print elements of a 1-D array.

```
#include<stdio.h>

int main()
{
    int arr[5], i;

    for(i = 0; i < 5; i++)
    {
        printf("Enter a[%d]: ", i);
```

```
10      scanf("%d", &arr[i]);
11    }
12
13    printf("\nPrinting elements of the array: \n\n");
14
15    for(i = 0; i < 5; i++)
16    {
17      printf("%d ", arr[i]);
18    }
19
20    // signal to operating system program ran fine
21    return 0;
22  }
```

Expected Output:

```
1  Enter a[0]: 11
2  Enter a[1]: 22
3  Enter a[2]: 34
4  Enter a[3]: 4
5  Enter a[4]: 34
6
7  Printing elements of the array:
8
9  11 22 34 4 34
```

How it works:

In Line 5, we have declared an array of `5` integers and variable `i` of type `int`. Then a for loop is used to enter five elements into an array. In `scanf()` we have used `&` operator (also known as the address of operator) on element `arr[i]` of an array, just like we had done with variables of type `int`, `float`, `char` etc. Line 13 prints `"Printing elements of the array"` to the console. The second for loop prints all the elements of an array one by one.

The following program prints the sum of elements of an array.

```
1   #include<stdio.h>
2
3   int main()
4   {
5     int arr[5], i, s = 0;
6
7     for(i = 0; i < 5; i++)
8     {
9       printf("Enter a[%d]: ", i);
10      scanf("%d", &arr[i]);
11    }
12
13    for(i = 0; i < 5; i++)
14    {
15      s += arr[i];
16    }
17
18    printf("\nSum of elements = %d ", s);
19
20    // signal to operating system program ran fine
21    return 0;
22  }
```

Expected Output:

```
1  Enter a[0]: 22
2  Enter a[1]: 33
3  Enter a[2]: 56
4  Enter a[3]: 73
5  Enter a[4]: 23
6
7  Sum of elements = 207
```

How it works:

The first for loop asks the user to enter five elements into the array. The second for loop reads all the elements of an array one by one and accumulate the sum of all the elements in the variable `s`. Note that it is necessary to initialize the variable `s` to `0`, otherwise, we will get the wrong answer because of the garbage value of `s`.

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: `datatype array_name[size] = { val1, val2, val3, ..... valN };`

`datatype` is the type of elements of an array.

`array_name` is the variable name, which must be any valid identifier.

`size` is the size of the array.

`val1`, `val2` … are the constants known as initializers. Each value is separated by a comma(,) and then there is a semi-colon (;) after the closing curly brace (}).
Here is are some examples:

```
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
```

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

```
1  float temp[] = {12.3, 4.1, 3.8, 9.5, 4.5}; // an array of 5 floats
2
3  int arr[] = {11, 22, 33, 44, 55, 66, 77, 88, 99}; // an array of 9 ints
```

If the number of initializers is less than the specified size then the remaining elements of the array are assigned a value of `0`.

```
1  float temp[5] = {12.3, 4.1};
```

here the size of `temp` array is `5` but there are only two initializers. After this initialization the elements of the array are as follows:
`temp[0]` is `12.3`
`temp[1]` is `4.1`
`temp[2]` is `0`
`temp[3]` is `0`
`temp[4]` is `0`
If the number of initializers is greater than the size of the array then, the compiler will report an error.

Derived Data Types (Structures and Unions)

Structure

The structure is a user-defined data type in C, which is used to store a collection of different kinds of data.

- The structure is something similar to an array; the only difference is array is used to store the same data types.
- *struct* keyword is used to declare the structure in C.
- Variables inside the structure are called *members of the structure*.

1. Defining a Structure in C

2. Accessing Structure Members in C

Defining a Structure in C

Syntax:

```
struct structureName

{

   //member definitions

};
```

Example:

```
struct Courses

{

   char  WebSite[50];

   char  Subject[50];

   int   Price;
```

```
};
#include<stdio.h>
#include<string.h>
struct Courses
{
   char  WebSite[50];
   char  Subject[50];
   int   Price;
};

void main( )
{
   struct Courses C;

   //Initialization
   strcpy( C.WebSite, "myweb.in");
   strcpy( C.Subject, "The C Programming Language");
   C.Price = 0;

   //Print
   printf( "WebSite : %s\n", C.WebSite);
   printf( "Book Author : %s\n", C.Subject);
   printf( "Book Price : %d\n", C.Price);
}
```

Program Output:

```
WebSite : myweb.in
```

C UNIONS

Unions are user-defined data type in C, which is used to store a collection of different kinds of data, just like a structure. However, with unions, you can only store information in one field at any one time.

- Unions are like structures except it used less memory.
- The keyword *union* is used to declare the union in C.
- Variables inside the union are called *members of the union*.

1. Defining a Union in C

2. Accessing Union Members in C

Defining a Union in C

Syntax:

```
union unionName
{
   //member definitions
};
```

Example:

```
union Courses
{
   char  WebSite[50];
   char  Subject[50];
   int   Price;
};
```

Accessing Union Members in C

Example:

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

union Courses
{
   char  WebSite[50];
   char  Subject[50];
   int   Price;
};

void main( )
{
   union Courses C;
```

```
   strcpy( C.WebSite, "myweb.in");
   printf( "WebSite : %s\n", C.WebSite);

   strcpy( C.Subject, "The C Programming Language");
   printf( "Book Author : %s\n", C.Subject);

   C.Price = 0;
   printf( "Book Price : %d\n", C.Price);
}
```

Program Output:

```
WebSite : myweb.in
```