Identifier

Identifiers refer to the names of variables, functions and arrays. They are user-defined names and consist of a sequence of letters and digits, with a letter as a first character. Both uppercase and lowercase letters are permitted. The underscore character is also permitted in identifiers.

Constants

Constants in C refer to fixed values that do not change during the execution of a program.

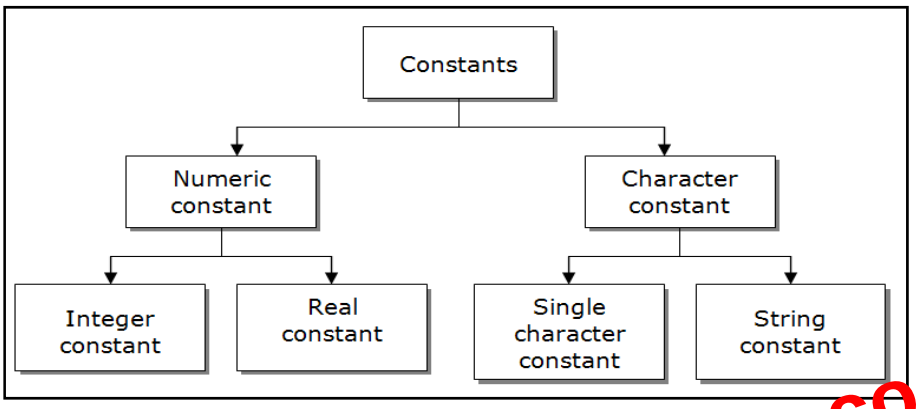
Types of C Constants

C constants can be divided into two major categories:

(a) Primary Constants

(b) Secondary Constants

These constants are further categorized as shown in Figure.



Integer Constants

An integer constant refers to a sequence of digits. There are three types integers, namely, ***decimal, octal, and hexa- decimal.***

Decimal Constant

A decimal integer constant consists of any combination of digits from the set 0 through 9. Eg: 123, -321etc.

Note: Embedded spaces, commas and non-digit characters are not permitted between digits. Eg: (1) 15 750 (2) $1000

Octal Constant

An octal integer constant consists of any combination of digits from the set 0 through 7, with a leading 0. Eg: 1) 037 2) 0435

Hexadecimal Constant

A sequence of digits preceded by 0x or 0X is considered as hexadecimal integer. They may also include 0-9 and alphabets A through F or a through f. Eg: 1) 0X2 2) 0x9F 3) 0Xbcd

Real Constants

Certain quantities that vary continuously, such as distances, heights etc., are represented by numbers containing functional parts like 17.548. Such numbers are called real (or floating point) constants. Eg: 0.0083, -0.75 etc., A real number may also be expressed in exponential or scientific notation. Eg: 215.65 may be written as 2.1565e2

Single Character Constants

A single character constant contains a single character enclosed within a pair of single quote marks. Eg: '5' - 'X' - ' - ; - ' - ,

Rules for Constructing Variable Names

(a) A variable name is any combination of 1 to 31 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 247 characters. Still, it would be safer to stick to the rule of 31 characters. Do not create unnecessarily long variable names as it adds to your typing effort.
(b) The first character in the variable name must be an alphabet or underscore.
(c) No commas or blanks are allowed within a variable name.
(d) No special symbol other than an underscore (as in gross_sal) can be used in a variable name.
Ex.: si_int , m_hra , pop_e_89

Syntax of data type declaration

Data_Type var1,var2,…….;

These rules remain same for all the types of primary and secondary variables.

Rules for C Program

Before we begin with our first C program do remember the following rules that are applicable to all C programs:

(a) Each instruction in a C program is written as a separate statement. Therefore a complete C program would comprise of a series of statements.
(b) The statements in a program must appear in the same order in which we wish them to be executed; unless of course the logic of the problem demands a deliberate 'jump' or transfer of control to a statement, which is out of sequence.
(c) Blank spaces may be inserted between two words to improve the readability of the statement. However, no blank spaces are allowed within a variable, constant or keyword.
(d) All statements are entered in small case letters.
(e) C has no specific rules for the position at which a statement is to be written. That's why it is often called a free-form language.
(f) Every C statement must end with a ;. Thus ; acts as a statement terminator.

C Instructions

Now that we have written a few programs let us look at the instructions that we used in these programs. There are basically three types of instructions in C:

(a) Type Declaration Instruction

(b) Arithmetic Instruction

(c) Control Instruction

The purpose of each of these instructions is given below:

(a) Type declaration instruction – To declare the type of variables used in a C program.
(b) Arithmetic instruction – To perform arithmetic operations between constants and variables.
(c) Control instruction – To control the sequence of execution of various statements in a C program

Data Types

C language provides various data types for holding different kinds of values. There are several integral data types, a character data type, floating point data types for holding real numbers and more. In addition you can define your own data types using aggregations of the native types.
C supports the four classes of data types

1. Primary (or fundamental) data types
2. User-defined data types
3. Derived data types
4. Empty data set

All c compiler support four fundamental data types, namely integer (**int**), character (**char**), floating point (**float**), and double – precision floating point (**double**).

Initializing Pointers

It is always good practice to initialize pointers as soon as it is declared. Since a pointer is just an address, if it is not initialized, it may randomly point to some location in memory. The ampersand (&) symbol, also called address operator, is applied to a variable to refer the address of that variable. Initializing pointers can be made to point to a variable using an assignment statement. The syntax is:

```
ptr_variable = &variable ;
```

Here, the address of the variable is assigned to ptr_variable as its value. For example:

```
ptr = &price ;
```

Will cause ptr to point to price i.e., ptr now contain the address of price. A pointer variable can be initialized in its declaration itself. For example:

```
int price, *ptr = &price ;
```

Is also valid.

Accessing a Variable through Its Pointer

Accessing the value of the variable using pointer is done by using unary operator * (asterisk), usually known as indirection operator. Consider the following statements:

```
int price, *ptr, n ;
price = 100 ;
ptr = &price ;
n = *ptr ;
```

The first line declares the price and n as integer variables and ptr as a pointer variable pointing to an integer. The second line assigns the value 100 to price and third line assigns the address of price to the pointer variable ptr. The fourth line contains the indirection operator *. When the operator * is placed before a pointer variable in an expression (on the right hand side of the equal sign), the pointer returns the value of the variable of which the pointer value is the address.
In this case, *ptr returns the value of the variable price, because ptr is the address of price. The * can be remembered as value at address. Thus the value of n would be 100.

Array

An array is a group of related data items that share a common name. Arrays are the contiguous memory location used to store similar data type or in other terms we can say Arrays are a data structure which holds multiple variables of the same data type.

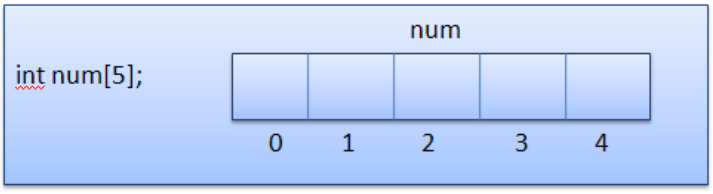
One Dimensional Array

An array with a single subscript is known as one dimensional array. Consider the case where a programmer needs to keep track of a number of people within an organization. So far, our initial attempt will be to create a specific variable for each user.
This might look like:

```
int name1 = 101;
int name2 = 232;
int name3 = 231;
```

It becomes increasingly more difficult to keep track of this as the number of variables increases. Arrays offer a solution to this problem. An array is a multi-element box, a bit like a filing cabinet, and uses an indexing system to find each variable stored within it. In C, indexing starts at zero. Arrays, like other variables in C, must be declared before they can be used. The replacement of the above example using arrays looks like:

```
int num[5];
```



Multi-Dimensional Array

Arrays of three or more dimension are called Multi-Dimensional Array.

General form Multi-Dimensional Array:

data_type **array_name[s1][s2][s3].....[sn];**

Example: **int survey[3][5][12]**

Here survey is a 3-dimensionalarray declared to contain 180 integer_type elements. (3x5x12=180)

Initialization of 4-Dimensional Array

static int **arr[3] [4] [2]={{{2,4}, {7,3}, {3,4}, {5,1}, }, {{3,4}, {3,4}, {3,2}, {4,5}}, {{2,3}, {2,7}, {2,3}, {4,3}}}**

In this example, the outer array has three element , each of which is a two dimensional array of four rows, each other of which is a one dimensional array of two elements.

String

A string is an array of characters. Strings in C are represented by arrays of characters. The end of the string is marked with a special character, the null character ('\0'), which is simply the character with the value 0.
Because C has no built-in facilities for manipulating entire arrays (copying them, comparing them, etc.), it also has very few built-in facilities for manipulating strings.

The operations that are performed on character strings are:

1. Reading and writing strings.
2. Combining strings together.
3. Copying one string to another.
4. Comparing strings for equality.
5. Extracting a portion of a string.

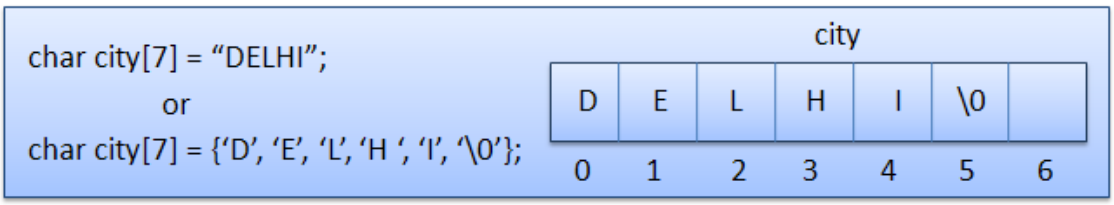
Decraring and Initializing String Variables

A string variable is any valid C variable name and is always declared as an array.
The general form of declaration of a string variable is

char name[30];

Therefore, the size should be equal to the maximum number of characters in the string plus one. Eg:

char city[7] = "DELHI"; or
char city[7] = {'D', 'E', 'L', 'H ', 'I', '\0'};



Reading Words

The familiar input function scanf() can be used with %s format specification to read in a string of characters.

Reading a Line of Text

It is not possible to use scanf() function to read a line containing more than one word. This is because the scanf() terminates reading as soon as a space is encountered in the input. We can use the getchar() function repeatedly to read single character from the terminal, using the function getchar(). Thus an entire line of text can be read and stored in an array.

Writing String To Screen

We have used extensively the *printf()* function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by the null character. For example *printf*("%s", name); can be used to display the entire contents of the array name.

Arithmetic Operations on Characters

C allows us to manipulate characters the same way we do with numbers. Whenever a character constant or character variable is used in an expression, it is automatically converted into integer value by the system. Eg: if the machine uses the ASCII representation, then,

```
x = 'a';
printf("%d \n",x);
```

will display the number 97 on the screen.

The C library supports a function that converts a string of digits into their integer values.

Standard Library String Functions

With every C compiler a large set of useful string handling library functions are provided. Figure lists he more commonly used functions along with their purpose.

Function	Use
strlen	Finds length of a string
strlwr	Converts a string to lowercase
strupr	Converts a string to uppercase
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strncmp	Compares first n characters of two strings
strcmpi	Compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp	Compares two strings without regard to case (identical to strcmpi)
strnicmp	Compares first n characters of two strings without regard to case
strdup	Duplicates a string
strchr	Finds first occurrence of a given character in a string
strrchr	Finds last occurrence of a given character in a string
strstr	Finds first occurrence of a given string in another string
strset	Sets all characters of string to a given character
strnset	Sets first n characters of a string to a given character
strrev	Reverses string

Putting String Together

Just as we cannot assign one string to another directly, we cannot join two strings together by the simple arithmetic addition. That is, the statements such as

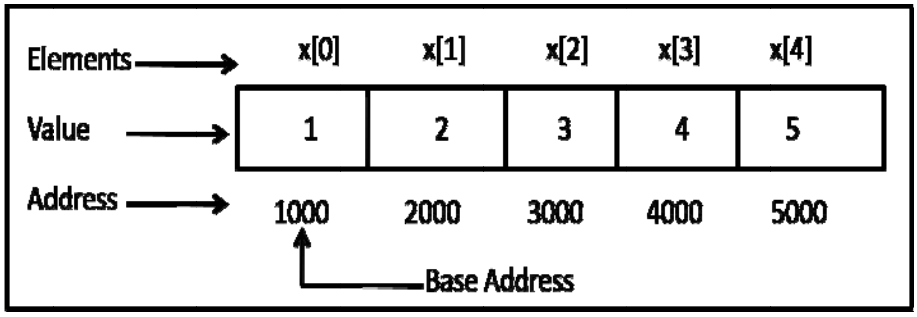
```
string3 = string1 + string2;
string2 = string1 + "hello";
```

Pointer and Array

When an array is declared, the compiler allocates a base address and sufficicient amout of storage to contain all the elemements of the array in contiguous memory locations. The base address is the location of the first element (index 0) of the array. Suppose we declare an array **x** as follows:

```
static int x[5] = {1,2,3,4,5};
```

Suppose the base address of **x** is 1000 and assuming that each integer requierd two bytes, the five elements will be stored as follows:



If we declare **p** as integer pointer, then we can make the pointer **p** to point to the array **x** by the following assignment:

```
p = x;
```

This is equivalent to

```
P = &x[0];
```

Now, we can access every value of **x** using p++ to move from one element to another. You can also use p --, p++ is used to jump to the next memory location, where p-- is used to jump previous memory location of array.

Pointer and Character String

In C, a constant character string always represents a pointer to that string. And therefore the following statements are valid:

```
char *name;
name="Delhi";
```

These statements will declare **name** as a pointer to character and assign to **name** the constant character string "Delhi". You might remember that this type of assignment does not apply to character array. The statement like

```
char name[20];
name="Delhi";
```

do not work

Structure

A structure is a convenient tool for handling a group of logically related data items, we can say, A structure contains a number of data types grouped together. These data types may or may not be of the same type.

For example, it can be used to represent a set of attributes, such as student _ name, roll _number and marks. The concept of a structure is analogous to that of a 'record' in many other languages. More examples of such structures are:

time:	seconds, minutes, hours
data:	day, month, year
book:	author, title, price, year
city:	name, country, population

Defining a Structure

Unlike arrays, structure must be defined first for their format that may be used later to declare structure variables.

The keyword *struct* declares a structure to hold the details of four data fields, namely title, author, pages, and price. These fields are called structure elements or members. Format of a structure definition is as follows:

```
struct tag_name(Structure_Name)
{
  data _ type member1;
  data _ type member2;
  --------- -----
  --------- -----
};
```

Array VS Structure

1) An array is a collection of related data elements of same type. Structure can have elements of different types.

2) An array is derived data type whereas structure is a programmer-defined one.

3) Any array behaves like a built-in data type. All we have to do is to declare an array variable and use it. But in the case of a structure, first we have to design and declare a data structure before the variables of that type are declared and used.

Structure Initialization

Like primary variables and arrays, structure variables can also be initialized when they are declared. The format used is quite similar to that used to initiate arrays.

```
struct book
{
        char name[10] ;
        float price ;
        int pages ;
} ;
struct book b1 = { "Basic", 130.00, 550 } ;
struct book b2 = { "Physics", 150.80, 800 } ;
```


Note the following points while declaring a structure type:

a) The closing brace in the structure type declaration must be followed by a semicolon.
b) It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the 'form' of the structure.
c) Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined. In very large programs they are usually put in a separate header file, and the file is included (using the preprocessor directive #include) in whichever program we want to use this structure type.

Accessing Structure Elements

Having declared the structure type and the structure variables, let us see how the elements of the structure can be accessed.

In arrays we can access individual elements of an array using a subscript. Structures use a different scheme. They use a dot (.) operator. So to refer to pages of the structure defined in our sample program we have to use,

b1.pages

Similarly, to refer to price we would use,

b1.price

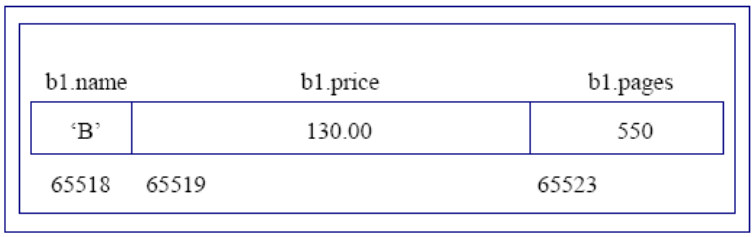
Note that before the dot there must always be a structure variable and after the dot there must always be a structure element.

How Structure Elements are Stored

Whatever be the elements of a structure, they are always stored in contiguous memory locations. The following example would illustrate this:

```
struct book
{
        char name ;
        float price ;
        int pages ;
} ;
struct book b1 = { 'B', 130.00, 550 } ;
```

Structure elements are stored in memory as shown in the Figure.



Coparision of Structure Variables

Two variables of the same structure type can be compared the same way as ordinary variables. If person1 and person2 belong to the same structure, then the following operations are valid:

Operation		Meaning
person1 = person2	-	Assign person2 to person1.
person1 = =person2	-	Compare all members of person1 and person2 and return 1 if they are equal, 0 otherwise.
person1 != person2	-	Return 1 if all the members are not equal, 0 otherwise.

Array of Structure

We use structure to describe the format of a number of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases, we may declare an array of structure, each elements of the array representing a structure variable.

***struct* class student[100];**

It defines an array called student, that consists of 100 elements. Each element is defined to be of the type struct class. Consider the following declaration:

```
struct marks
{
        int subject1;
        int subject2;
        int subject3;
};
main()
{
 static struct marks student[3] = { {45, 68, 81}, {75, 53, 69}, {57,36,71}};
```

//This declares the student as an array of three elements students[0], student[1], and student[2] and initializes their members as follows:

```
 student[0].subject1=45;
 student[0].subject2=68;
 ..............
student[2].subject3=71;
}
```

An array of structures is stored inside the memory in the same way as a multidimensional array.