

## • Chapter 09 : Elements of C

### 9.1 Overview

Q : 09-01-01 : Describe The Basic Elements (Building Blocks) of C Language ?

**Answer :**

**The Basic Elements (Building Blocks) of C Language :** Computer does not do anything on its own. The most important skill to learn is how intelligently one can program it. It can be used to solve multidimensional problems. The C being a magic tool for writing programs is used to solve variety of problems; a beginner just need practice and more practice of writing programs to master it. The proper use of basic elements (building blocks) of C language helps a lot to write effective C programs.

**Identifiers :** These are the names used to represent variables, constants, types, functions, and labels in the program. Identifiers in C can contain any number of characters, but only the first 31 are significant to C compiler. There are two types of identifiers in C :

**Standard Identifiers :** Like reserved words, standard identifiers have special meanings in C, but these can be redefined to use in the program for other purposes, however this practice is not recommended. If a standard identifier is redefined, C no longer remains able to use it for its original purpose. Examples of standard identifiers include printf and scanf, which are names of input / output functions defined in standard input / output library (stdio.h).

**User-Defined Identifiers :** In a C program, the programmer may need to access memory locations for storing data and program results. For this purpose memory cells are named that are called user-defined identifiers.

**Important Note :** C is a case sensitive language This means that C compiler considers uppercase and lowercase letters to be distinct characters. For example, the compiler considers SQUARE\_AREA and Square\_Area as two different identifiers referring to different memory locations.

### 9.2 Keywords

Q : 09-02-01 : Describe Keywords ?

**Answer :**

**Keywords :** Keywords or reserved words are the words, which have predefined meanings C. There are 32 words defined as keywords in C. These have predefined uses and cannot be used or redefined for any other purpose in a C program. They are used by the compiler as an aid to compile the program. They are always written in lower case. A complete list of ANSI C keywords is :

<a href="#"><u>auto</u></a>	<a href="#"><u>break</u></a>	<a href="#"><u>case</u></a>	<a href="#"><u>char</u></a>	<a href="#"><u>const</u></a>	<a href="#"><u>continue</u></a>	<a href="#"><u>default</u></a>
<a href="#"><u>do</u></a>						
<a href="#"><u>double</u></a>	<a href="#"><u>else</u></a>	<a href="#"><u>enum</u></a>	<a href="#"><u>extern</u></a>	<a href="#"><u>float</u></a>	<a href="#"><u>for</u></a>	<a href="#"><u>goto</u></a>
<a href="#"><u>if</u></a>						

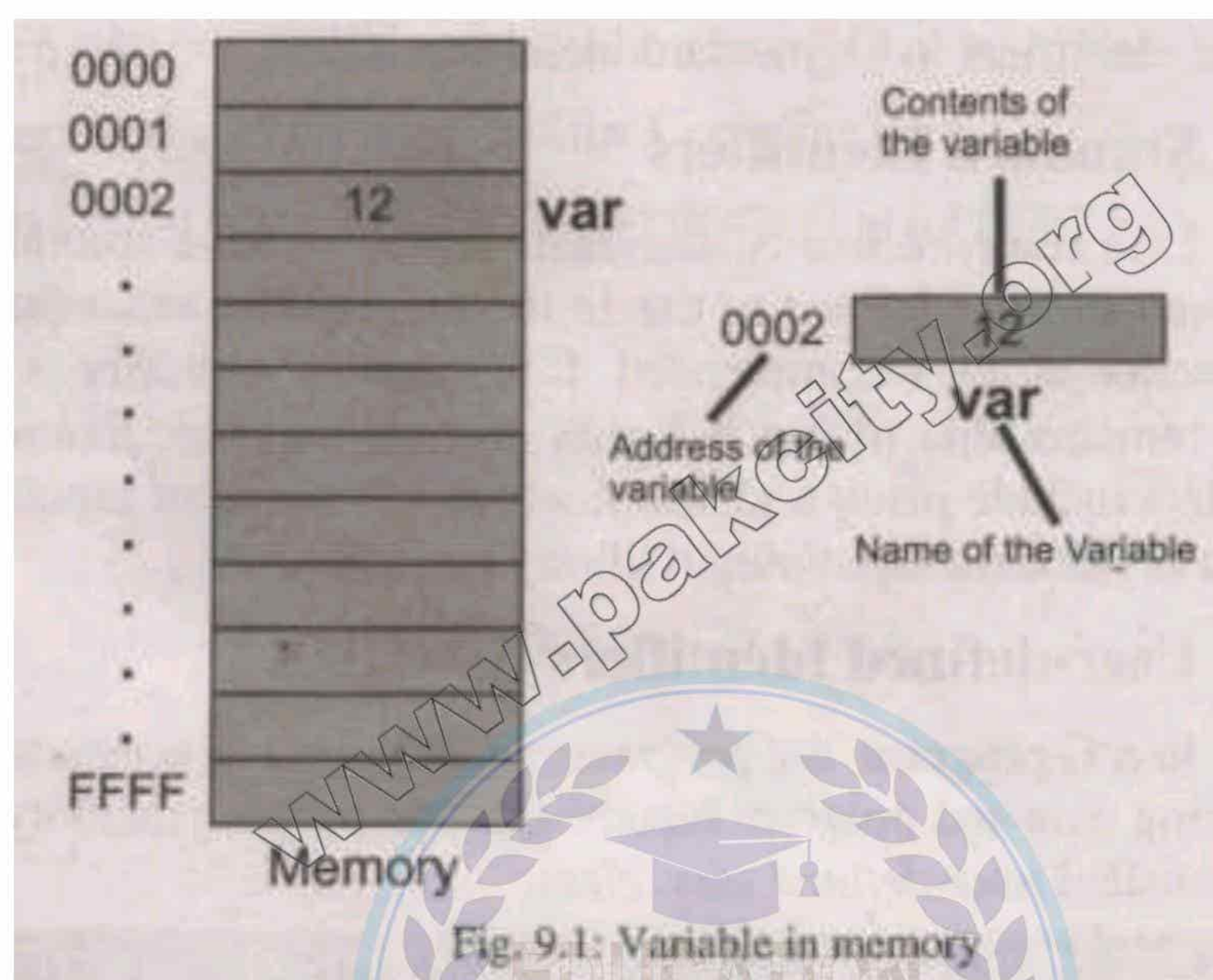


int                      long                      register                      return                      short                      signed                      sizeof  
static  
struct                      switch                      typedef                      union                      unsigned                      void                      volatile  
while

Q : 09-02-02 : Describe Variables ?

**Answer :**

**Variables :** Variables are named memory locations (memory cells), which are used to store program's input data and its computational results during program execution. As the name suggests, the value of a variable may change during the program execution. We are familiar with the concept of variable with reference to algebra. The variables are created in memory (RAM); therefore the data is stored in them temporarily. One should not mix the contents of a variable with its address. These are entirely different concepts. The contents of a variable can be thought of as the residents of your neighboring house, while address of the variable can be thought of as the address of that house.



Q : 09-02-03 : How do you declare Variables in C ?

**Answer :**

**Declaring a Variable, in C :** C is a strongly typed language i.e. all variables must be declared before being used. The compiler will report an error if an undeclared variable is in a program. A variable is declared in C by specifying its type (data type) and name. The syntax takes the form :

Data\_Type                      Variable\_Name ;

Data type refers to the type of the data being stored in the variable. For example :

int                      kgs ;  
double                      length ;

A list of names of variables separated by commas is specified with the data type to declare more variables of the same data type such as :

int                      marks, total\_students, no\_of\_rooms ;  
double                      kgs, length, volume, height ;



Q : 09-02-04 : Differentiate between Declaring vs Defining a Variable in C ?

**Answer :**

**Declaring vs Defining a Variable :** Variable declaration tells the compiler the name of the variable to be used in the program and the type of information stored in it. In a C program, the :

```
int          volume ;
char         ch ;
```

variable declarations tell the compiler the name of two variables (volume and ch) used to store an integer and character data respectively.

**Important Note 1 :** A variable declaration does not set aside memory location for the data to be stored. It just informs the compiler the name of the variable and the type of data to be stored in it, while the definition of the variable that set aside memory location for the variable.

**Important Note 2 :** However in C, the variable declaration statement not only declares the variable but also defines it as in case of above two statements. It does not mean that the declaration of a variable can not be separated from its definition in C.

Q : 09-02-05 : Describe Initializing a Variable ?

**Answer :**

**Initializing a Variable :** Assigning a value to a variable at the time of its declaration is called initializing a variable. In a C program, when a variable is declared, the compiler sets aside some memory space for it. This allocated memory space may contain data meaningless (garbage) to the program. The computations involving this variable may produce unexpected results. To avoid this situation, all variables should be declared and initialized before being used. In C, a variable can also be initialized at the time of its declaration e.g.,

```
int count ; // (Variable declaration and definition)
count = 125 ; // (Variable Initialization)
char ch = 'z' ; // (Variable declaration, definition and
Initialization)
float weight = 75.8 ;
```

Q : 09-02-06 : Describe Rules for Naming Variables in C ?

**Answer :**

**Rules for Naming Variables in C :** To use variables in C programs, we must know the rules to choose names for them. Here are some important rules for naming a variable in C :

**Allowed Symbols / Characters :** A variable name can consists of letters, digits, and the underscore character(\_).

**The First Symbol / Character :** The first character of the name must be a letter or underscore, but use of underscore is not recommended. The second character onwards digits / numeric literals (0 .. 9) are also legal.

The Choices at First Character :      A .. Z,  
   a .. z,  
   Underscore ( \_ )



The Choices at Later Characters :

Total 53.  
 A .. Z,  
 a .. z,  
 Underscore ( \_ )  
 0 .. 9  
 Total 63.

Thus 9winner, #home, and 2kgms are invalid names in C.

**C is Case Sensitive Language** : C is a Case Sensitive Language, thus, the names count, Count and COUNT refer to three different variables.

**Keywords** : C Language keywords can't be used as variable names e.g., we can not use int, void, signed, or while as variable names.

**Length** : For many compilers, a C variable name can be up to 31 characters long. (It can actually be longer than that i.e., 255 but the first 31 characters of the name are significant) e.g., Turbo C++ restricts the maximum length of a variable name to 31 characters. Hence, the names problem\_solving\_techniques\_in\_C (31 characters) and problem\_solving\_techniques\_in\_C\_language (40 characters) would appear to be the same to the compiler. The compiler does not differentiate these two names because the first 31 characters of both are the same.

**Blank Space** : Blank spaces are not allowed in the name e.g., (problem solving) is an invalid variable name in C.

**Only One Data Type** : A variable can only be declared for one data type.

**Important Note** : C programmers commonly use only lowercase letters in variable names, although this isn't required. Using all-uppercase letters is usually reserved for the names of constants.

Q : 09-02-07 : Explain, Why Variable Names should be Readable ?

**Answer :**

**Variable Names should be Readable** : Let's consider a program that calculates loan payments, could store the value of the prime interest rate in a variable named interest\_rate. The variable name helps make its usage clear. We could also create a variable named x or even Ahmed; it doesn't matter to the C compiler. Many naming conventions are used for variable names. We've seen one style: interest\_rate. Using an underscore to separate words in a variable name makes it easy to interpret. Another style is to capitalize the first letter of each word. Instead of interest\_rate, the variable would be named as InterestRate or interestRate.

## 9.4 Constants

Q : 09-04-01 : Explain, Constants in C Language ?

**Answer :**

**Constants** : A constant is a quantity whose value can not be changed. Unlike a variable, the value stored in a constant can't be changed during program execution.

The define directive can be used to define constant macros : `#define PI 3.142857 .`

Defines a constant i.e. PI, whose value will remain unchanged during the program execution. C has two types of constants; numeric constants and character constants, each with its own specific uses :



**Numeric Constants** : Numeric constants consist of numbers. It can be further divided into integer and floating point constants. Integer constants represent values that are counted and do not have a fractional part e.g., +56, -678, 8, etc. Floating point constants represent values that are measured e.g., -4.786, 5.0, 0.45 etc.

**Character Constants** : A character constant is a single alphabet, a single digit or a single symbol enclosed within apostrophes e.g., 'A' is a valid character constant. e.g., 'A', 'I', '5', '=' etc. The maximum length of a character constant is 1 character.

**ASCII Character Set** : The set of ASCII (American Standard Code for Information Interchange) Characters consists of control, printable and extended which is 0 to 255.

## List of ASCII Codes

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
0	null	37	%	74	J	111	o	148	ö	185	¶	222	¶
1	start of heading	38	&	75	K	112	p	149	ò	186	¶	223	¶
2	start of text	39	'	76	L	113	q	150	û	187	¶	224	¶
3	end of text	40	(	77	M	114	r	151	ù	188	¶	225	¶
4	end of transmission	41	)	78	N	115	s	152	ÿ	189	¶	226	¶
5	inquiry	42	*	79	O	116	t	153	Ö	190	¶	227	¶
6	acknowledge	43	+	80	P	117	u	154	Ü	191	¶	228	¶
7	bell	44	,	81	Q	118	v	155	ø	192	¶	229	¶
8	backspace	45	-	82	R	119	w	156	£	193	¶	230	¶
9	horizontal tab	46	.	83	S	120	x	157	¥	194	¶	231	¶
10	line feed/new line	47	/	84	T	121	y	158	₣	195	¶	232	¶
11	vertical tab	48	0	85	U	122	z	159	f	196	—	233	¶
12	form feed/new page	49	1	86	V	123	{	160	á	197	+	234	¶
13	carriage return	50	2	87	w	124		161	í	198	†	235	¶
14	shift out	51	3	88	X	125	}	162	ó	199	‡	236	¶
15	shift in	52	4	89	Y	126	~	163	ú	200	‡	237	¶
16	data link escape	53	5	90	Z	127	DEL	164	ñ	201	‡	238	¶
17	device control 1	54	6	91	[	128	Ç	165	Ñ	202	‡	239	¶
18	device control 2	55	7	92	\	129	ü	166	ª	203	‡	240	¶
19	device control 3	56	8	93	]	130	é	167	º	204	‡	241	¶
20	device control 4	57	9	94	^	131	â	168	¿	205	=	242	¶
21	negative acknowledge	58	:	95	_	132	ä	169	ƒ	206	‡	243	¶
22	synchronous idle	59	;	96	`	133	à	170	¬	207	‡	244	¶
23	end of transmission block	60	<	97	a	134	å	171	½	208	‡	245	¶
24	cancel	61	=	98	b	135	ç	172	¼	209	‡	246	¶
25	end of medium	62	>	99	c	136	ê	173	ï	210	π	247	¶



26	substitute	63	?	100	d	137	ë	174	«	211	ℓ	24
27	escape	64	@	101	e	138	è	175	»	212	Ô	24
28	file separator	65	A	102	f	139	ï	176	⋮	213	ƒ	25
29	group separator	66	B	103	g	140	î	177	⋮	214	π	25
30	record separator	67	C	104	h	141	ì	178	⋮	215	‡	25
31	unit separator	68	D	105	i	142	Ä	179		216	≠	25
32	space	69	E	106	j	143	Å	180	└	217	┘	25
33	!	70	F	107	k	144	É	181	≠	218	┘	25
34	"	71	G	108	l	145	æ	182	≠	219	■	
35	#	72	H	109	m	146	Æ	183	π	220	■	
36	\$	73	I	110	n	147	ô	184	≠	221	■	

## 9.5 Data Type

Q : 09-05-01 : Explain, Data Types in C Language ?

**Answer :**

**Data Types :** In C, the data type defines a set of values and a set of operations on those values. We know that computer program manipulates various types of data. The data is given to the program as input. The data is processed according to the program instruction and output is returned. In program designing, the data and its types are defined before designing the actual program used to process the data. The type of each data value is identified at the beginning of program design. The values used in a program may be of different types.

**Association :** In a C program, we may need to process different type of data. Therefore, variables should be capable of storing data of different types. This is done by associating certain data types to the variables.

**Pre-Defined and User-Defined Data Types :** To work with different types of data, C defines some standard data types and also allows us to define our own data types known as user-define data types. In C, a standard data type is one that is predefined in the language such as int, double, char, float etc. The important standard data types are :

**Data Types for Integers (int, short, long) :** In C, the data type int is used to represent integers - the whole numbers. This means that int variables store number that have no fractional parts such as +1128, -1010, 32432 etc. Along with standard type int, the compiler also supports two more types of integer i.e. short int and long int, often abbreviated to just short and long. In addition to these types, an integer variable can be signed or unsigned. If not mentioned, all integer variables are considered to be signed. The data types signed int and int can handle both signed and unsigned whole numbers such as 245, 1010, and -232 etc, whereas the data type unsigned can not handle negative numbers. Because of the limited size of the memory cell, all integers can not be represented by int, short or long.

Data Type	Other Name	Bytes	Range		Range	
Short	Short Int	2	-2 <sup>15</sup>	2 <sup>15</sup> - 1	-32768	
Unsigned Short	Unsigned Short Int	2	0	2 <sup>16</sup> - 1	0	
Int	Signed	2	-2 <sup>15</sup>	2 <sup>15</sup> - 1	-32768	



Unsigned Int	Unsigned	2	0	$2^{16} - 1$	0	
Long	Long Int	4	$-2^{31}$	$2^{31} - 1$	-2147483648	21474
Unsigned Long	Unsigned Long Int	4	0	$2^{32} - 1$	0	42949

When a variable of type int is declared, the compiler allocates two bytes of memory to it. Therefore, only the numbers ranging from  $2^{15}$  through  $2^{15} - 1$  (i.e. -32768 to 32767) can be represented with the int type variables. The variable of type unsigned int can handle numbers ranging from 0 through  $2^{16} - 1$  (i.e. 0 to 65535). The long is used to represent larger integers. It occupies four bytes of memory and can hold numbers ranging from  $2^{31}$  through  $2^{31} - 1$  (i.e. -2147483648 to 2147483647).

### **Data Types for Floating Point Numbers (float, double, long double) :**

Floating point numbers are the numbers that have a fractional part e.g. 12.35874, 0.54789, and -8.64, 101.003 etc. The floating point numbers are represented in computer in a format that is analogous to scientific notation (floating-point format). The storage area occupied by the number is divided into two sections: the mantissa and the exponent. Mantissa is the value of the number and the exponent is the power to which it is raised. For example, in exponential notation the number 245634 would be represented as  $2.45634 \times 10^5$ , where 2.45634 is the mantissa and 5 is the exponent. However in C, the representation of scientific notation is slightly different e.g. the above number will be represented as 2.45634e5. We don't express the exponent as the power of 10, rather the letter e or E is used to separate exponent from the mantissa. If the number is smaller than 1 (one), then exponent would be negative. For example, the number 0.00524 will be represented in computer as 5.24E-3. ANSI C specifies three floating point types that differ in their memory requirements : float, double, and long double. These data types provide higher precision than the data types used for integers.

Data Type	Bytes	Precision	Range	
Float	4	6	$10^{-38}$	$10^{38}$
Double	8	15	$10^{-308}$	$10^{308}$
Long double	10	19	$10^{-4932}$	$10^{4932}$

**Working with Floating Point Numbers :** While working with floating point numbers, may encounter some problems. For example, manipulation of very large and very small floating point numbers may show unexpected results. When we add a large number and a small number, the larger number may cancel out the smaller number; resulting in a cancellation error e.g. the result of addition of 1970.0 and 0.000000 1243 may compute to 1970.000000 on some computers. When two very small numbers are manipulated, the result may be too small to be represented accurately, so it will be represented as zero. This phenomenon is called arithmetic UNDERFLOW. Similarly, manipulation of two very large numbers may result to a too large number to be represented. This phenomenon is call arithmetic OVERFLOW.

**Data Type for Characters - char :** The data type char is used to represent a letter, number, or punctuation mark (plus a few other symbols). A char type



variable occupies 1 byte in memory and can represent individual characters such as 'a', 'x', '5' and '#' etc. The character '5' is manipulated quite differently than the integer 5 in the computer, so one should not consider both the same. In C, a character is expressed as enclosed in apostrophes such as 'o', and 'u' etc. Like numbers, characters can also be compared, added and subtracted. Let's look at the following programs to understand the concept :

```
#include<stdio.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    '2'   ;
    ch2   =    '6'   ;
    sum   =    ch1   +    ch2   ;
    printf("Sum = %d", sum);
}
```

Program Output :

Sum = 104



```
#include<stdio.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    '2'   ;
    ch2   =    '6'   ;
    sum   =    ch1   +    ch2   ;
    printf("Sum = %d", sum);
    printf("Sum = %c", sum);
}
```

Program Output :

Sum = 104 Sum = h

```
#include<stdio.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    '2'   ;
    ch2   =    '6'   ;
    sum   =    ch1   +    ch2   ;
    printf("Sum = %d and Symbol = %c", sum,
    sum) ;
}
```

Program Output :

Sum = 104 and Symbol = h



```

#include<stdio.h>
void main(void)
{
    char  ch1,  ch2,  sum  ;
    ch1   =    2    ;
    ch2   =    6    ;
    sum   =   ch1  +   ch2  ;
    printf("Sum = %d and Symbol = %c", sum,
    sum) ;
}

```

Program Output :

Sum = 8 and Symbol =

In fact, characters are stored in the form of ASCII (American Standard Code for Information Interchange) code. When we add, subtract or compare two characters, then instead of characters their ASCII codes are manipulated. above example, because the ASCII codes of characters '2' and '6' are 50 and 54 respectively therefore the sum is 104. In ASCII, the printable characters have codes from 32 (code for a blank or space) to 126 (code for symbol ~). The other codes represent nonprintable control characters. The signed and unsigned keywords can be used with char like they can with int. Signed characters can represent numbers ranging from -128 though 127, while unsigned characters can represent numbers from 0 to 255. English alphabets, numbers and punctuation marks are always represented with positive numbers.

## 9.6 Operators in C

Q : 09-06-01 : Explain, Data Types in C Language ?

**Answer :**

**Operators :** [Operators are symbols, which are used to perform certain operations on data]. C is equipped with a rich set of operators. These include arithmetic operators, relational operators, logical operators, bitwise operators, and many others :

**Arithmetic Operators :** Arithmetic operators are used to perform arithmetic operations values (numbers). The C language incorporates the following standard arithmetic operators :

The use of first four operators is straightforward. The last operator is modulus (also called remainder operator). Contrary to the division operator, which returns the quotient, it returns the remainder of an integral division. For example, if a, and b are two integers having values 8 and 3 respectively, then the express  $a \% b$  will be evaluated to 2, which is the remainder of integral division.



Operation	Symbol	Algebraic Expression	Expressions in C
Addition	+	$A + b$	$a + b$
Subtraction	-	$A - b$	$a - b$
Multiplication	*	$A \times b$	$a * b$
Division	/	$A / b$	$a / b$
Modulus	%	$a \text{ mod } b$	$a \% b$

**Relational Operators :** Relational operators are used to compare two values. These operators always evaluates to true or false. They produce a non-zero value (in most cases 1) if the relationship evaluates to true and a 0 if the relationship evaluates to false. The six basic relational operators in C are : Suppose a, b and c are three integer variables having values 123, 215 and 123 respectively then :

Operation	Symbol	Expression	Evaluation
Equal to (comparison)	==	$a == c$	true (non-zero)
Less than	<	$b < a$	false (zero)
Greater than	>	$a > c$	false (zero)
Less than or Equal to	<=	$a <= b$	true (non-zero)
Greater than or Equal to	>=	$b <= a$	false (zero)
Not Equal to	!=	$a != b$	true (non-zero)

**Logical Operators :** Logical operators combine two or more relational expressions to construct compound expressions. The logical operators are && (logical AND), || (logical OR), and ! (logical NOT). The first logical operator && (logical AND) when combines two conditions, evaluates to TRUE if both the conditions are TRUE, otherwise it evaluates to FALSE. The second logical operator || (logical OR) when combines two conditions, evaluates to TRUE if any one of the conditions evaluates to TRUE, otherwise evaluates to FALSE. Similarly, the third logical operator (logical NOT) when applied to a condition, reverse the result of; the evaluation of the condition, this means that if the condition evaluates to TRUE, the logical NOT operator evaluates to FALSE and vice versa. The three logical operators can take the following general forms :

exp1 && exp2  
 exp1 || exp2  
 !(expression)

**Assignment Operator :** In addition to basic C operators (arithmetic, logical, and relational) there are some other important operators, which one should know to write C program. These includes assignment operator, increment and decrement operators. The assignment operator is used to store a value or a computational result in a variable. In C, the symbol = represents the assignment operator e.g. in the following statement, values of the two variables, height and width, multiplied and the result is assigned to the variable Area.  $\text{Area} = \text{height} * \text{width}$  ;

The value to the right side of the operator is assigned to the variable on the left side of the assignment operator. This statement is also called assignment statement.



**Assignment Statement** : The assignment statement takes the general form :

variable = expression

The expression on the right side of the operator is evaluated first and the result is then assigned to the variable on the left side of the operator. The expression can be a variable, a constant or arithmetic, relational or logical expression.

**Important Note** : Writing variable to the right side and the expression to the left side of assignment operator will cause a syntax error.

**Increment and Decrement Operators** : The increment operator increases the value of its operand by one. It is denoted by the symbol ++ e.g. count++, where count is a variable. The effect of this expression is equivalent to the following expression :

count = count + 1 ;

When ++ precedes its operand, it is called prefix increment. When the ++ follows its operand, it is called postfix increment. Consider the following statements :

j        =        10 ;  
i        =        ++j ;

The first statement assigns the value of 10 to the variable j. In the second statement, first the value of j will be incremented by one (i.e. the value of j will become 11) and then the result will be assigned to the variable i. Hence, the variable j will be assigned the value of eleven (11). Therefore, after execution of the two statements, both the variables will have the same value i.e. eleven (11). Now, consider the following statements :

j        =        10 ;  
i        =        j++ ;

The execution of the first statement will take place as in above case. In the second statement, first the value of j (i.e. 10) will be assigned to the variable i, and then the value of j will be incremented by one. Hence, the variable i will be assigned the value of 10. Therefore, after execution of the two statements, the variable j will have the value of 11 and the variable i will have the value of 10. C also provides a decrement operator denoted by the symbol -- e.g. count--. The effect of this expression is equivalent to the following expression :

count = count - 1 ;


It can be used as either the prefix operator or postfix operator in the same way as the increment operator is used.

**Compound Assignment Operators** : The ++ and -- operators respectively increment and decrement the value of their operand by one. There are five other compound assignment operators that can increment or decrement the value of their operand by other than one. These are +=, -=, \*=, /= and %= operators. For example, the statement j += 5 increases the value of j by 5 and the statement j -= 5 decreases the value of j by 5. Similarly, we can also use the other operators in the same way.



<pre>j = 10; j *= 5;</pre> <p><u>After Execution of statements:</u></p> <ul style="list-style-type: none"> <li>The value of j will be 50</li> </ul>	<pre>j = 10; j /= 5;</pre> <p><u>After Execution of statements:</u></p> <ul style="list-style-type: none"> <li>The value of j will be 2</li> </ul>
---	--

**Operator Precedence :** An operator's precedence determines its order of evaluation in an expression.

Operator	Precedence
! (logical NOT)	Highest  Lowest
*, /, %	
+, -	
<, <=, >, >=	
==, !=	
&&	
= (assignment operator)	

**Important Note :** The logical NOT operator has the highest priority. It is unary operator - unary operators are those operators that have just one operand. Then comes arithmetic operators, relational operators, logical AND, logical OR and the assignment operators, which are all binary operators - binary operators are those operators that have two operands.

## 9.7 Expression

Q : 09-07-01 : Define and Explain Expression ?

**Answer :**

**Expression :** [An expression is the combination of operators and operands. The operand may either be a constant or a variable e.g.,  $a+b$ ,  $7+m$  etc]. [An expression, in which only arithmetic operators operate on operands, is known as arithmetic expression]. To solve different mathematical problems, one needs to write arithmetic expressions. Arithmetic expressions involve integers and floating point numbers, which are manipulated with arithmetic operators.

**Data Type of an Expression :** The data type of expression (in fact, the data type of the result of expression) depends on the types of its operand. For example, consider the type of expression involving int or double type operands is of the form :

expr1 operator expr2

If both the operands are of type int, the result of the expression will be evaluated to an int type value. However, in case of mixed-type expression, one must be careful. A mixed-type expression is one in which operands are of different data types e.g. if the



type of operand1 is int and the type of operand2 is double then the expression will always be evaluated to a double type value.

Arithmetic Operator	Examples
+	2 + 3 is 5 2.0 + 3.0 is 5.0
-	3 - 2 is 1 3.0 - 2.0 is 1.0
*	2 * 3 is 6 2.0 * 3.0 is 6.0
/	5 / 2 is 2 (integral division, because both the operands are integers) 5.0 / 2.0 is 2.5
%	5 / 2 is 2

5 % 2 is 1

**Working With Division Operator :** The manipulation of division operation is slightly different from other arithmetic operations in C. One should be careful while dividing number, as the result of division of two integers may not be an integer. C handles the division intelligently. When the divisor and the dividend both are integers, the fractional part of the quotient (if exists) is truncated. For example the value of 7.0/2.0 is 3.5 but the value of 7/2 is the integral part of the result i.e. 3. Similarly, the value of 198.0/100.0 is 1.98, but the value of 198/100 is the integral part only i.e. 1. That's how C performs the division operation. So, to get the accurate result, at least one floating point number should be involved in division operation. Otherwise, integral division will take place and we will get the integral value.

When a type double expression or value is assigned to a type int variable, the fractional part is truncated since it can not be represented in int type variable. For example, let a and b be two variables of type double and int respectively.

```
a = 5 * 0.5 ;
b = 5 * 0.5 ;
```

Now a becomes 2.5 and b becomes 2.

## 9.8 Comments

Q : 09-07-01 : Explain importance, usage and style of Comments in C?

**Answer :**

**Comments :** Comments are used to increase the readability of the program. With comments, informative notes are inserted in the program's code, which help in debugging and modifying the program. In C, there are two ways to comment the code.

**Single-Line Comments :** One can insert single line comments by typing two (forward) slashes (//) at the start of the note :

```
// This program calculates the factorial of the given number
```

**Multi-Line Comments :** Multi-line comments are used to add such informative notes

which extend to multiple lines. Multi-line comments can be inserted to the code (program code) by placing (/\*) characters at the beginning of the comments



(informative note), this is called opening of the multi-line comments. Each multi-line comment must end with letters (\*/) Omitting ending letters (\*/) will cause the whole program code beneath the opening letters (/\*) for comments to be commented.

```
/*  
  This program prints a single line message  
  Author:   Student  
  Date:    14-05-2005  
*/  
void main (void)  
{  
    // the following line of code prints a message  
    printf(" This is my first program");  
}
```

**Important Note :** Comments are completely ignored by the compiler while generating object code.

