C - Language Overview

C is a general purpose high level language that was originally developed by Dennis M. Ritchie to develop the Unix operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The UNIX operating system, the C compiler, and essentially all UNIX applications programs have been written in C. The C has now become a widely used professional language for various reasons.

- Easy to learn
- Structured language
- It produces efficient programs.
- It can handle low-level activities.
- It can be compiled on a variety of computer platforms.

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around 1970
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- The UNIX OS was totally written in C By 1973.
- Today C is the most widely used and popular System Programming Language.
- Most of the state of the art softwar's have been implemented using C.
- Today's most popular Linux OS and RBDMS MySQL have been written in C.

Why to use C ?

C was initially used for system development work, in particular the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Data Bases
- Language Interpreters
- Utilities

C - Environment Setup

Before you start doing programming using C programming language, you need following two software's available on your computer, (a) Text Editor and (b) The C Compiler.

Text Editor

This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi

Name and version of text editor can vary on different operating systems. For example Notepad will be used on Windows and vim or vi can be used on windows as well as Linux, or Unix.

The files you create with your editor are called source files and contain program source code. The source files for C programs are typically named with the extension **.c**.

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, compile it and finally execute it.

The C Compiler

The source code written in source file is the human readable source for your program. It needs to be "compiled", to turn into machine language so that your CPU can actually execute the program as per instructions given.

This C programming language compiler will be used to compile your source code into final executable program. I assume you have basic knowledge about a programming language compiler.

Most frequently used and free available compiler is GNU C/C++ compiler or Turbo C++

C - Program Structure

Before we study basic building blocks of the C programming language, let us look a bare minimum C program structure so that we can take it as a reference in upcoming chapters.

A C program basically consists of the following parts:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

C Hello World Example

Let us look at a simple code that would print the words "Hello World":

```
#include <stdio.h>

void main()
{
   /* my first program in C */
   printf("Hello, World! \n");
c=a*b;

   return;
}
```

First.c

Let us look various parts of the above program:

1. The first line of the program *#include <stdio.h>* is a preprocessor command which tells a C compiler to include stdio.h file before going to actual compilation.
2. The next line *int main()* is the main function where program execution begins.
3. The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
4. The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
5. The next line **return 0;** terminates main()function and returns the value 0.

Compile & Execute C Program:

Let's look at how to save the source code in a file, and how to compile and run it. Following are the simple steps:

1. Create a new file and Open a text editor and add the above mentioned code.
2. Save the file as *hello.c*
3. Compile the program using Compile menu or Alt+F9
4. If there is any error go back to the program, rectify it and re-compile again.
5. It it is error free, run or execute the program using Run menu or Ctrl+F9
6. Finally, You will be able to see *"Hello World"* printed on the screen

C - Basic Syntax

Tokens in C

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens:

```
printf("Hello, World! \n");
```

The individual tokens are:

```
printf
(
"Hello, World! \n"
)
;
```

Semicolons ;

In C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

For example, following are two different statements:

```
printf("Hello, World! \n");
return 0;
```

Comments

Comments are like helping text in your C program and they are ignored by the compiler. They start with /* and terminates with the characters */ as shown below for multiple lines.

```
/* my first program in C
carried out on 18th july 2012 at
VIT lab*/
```

For single line, use the comment line as shown below.

```
//my first program in C carried out on 18th july 2012 at VIT lab
```

You can not have comments with in comments and they do not occur within a string or character literals.

Identifiers

A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9).

C does not allow punctuation characters such as @, $, and % within identifiers. C is a **case sensitive** programming language. Thus *Manpower* and *manpower* are two different identifiers in C. Here are some examples of acceptable identifiers:

```
mohd      zara    abc   move_name  a_123
myname50  _temp   j     a23b9      retVal
```

Keywords

The following list shows the reserved words in C. These reserved words may not be used as constant or variable or any other identifier names.

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

Whitespace in C

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement:

int age;

There must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them. On the other hand, in the following statement

fruit = apples + oranges; // get the total fruit

No whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

C - Data Types

In the C programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows:

S.N.	Types and Description
1	**Basic Types:** They are arithmetic types and consists of the two types: (a) integer types and (b) floating-point

	types.
2	**Enumerated types:** They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
3	**The type void:** The type specifier *void* indicates that no value is available.
4	**Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred to collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see basic types in the following section where as other types will be covered in the upcoming chapters.

Integer Types

Following table gives you detail about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Size and Range of Data Types in C

Size and Range of data types in C. The size is calculated using sizeof(). The range of data types can be found by manually or using <limits.h> and <float.h>

Another factor on which the size of data type depends is the compiler on which you perform any program i.e. In turbo c/c++ the size of int is 2 bytes but in the compiler like code blocks, dev c/c++ e.t.c is 4 bytes.

For finding the size we need a sizeof() function defined under stdio.h. **sizeof()** function find the size in bytes. 0 or 1 takes 1 bit space. 1 byte = 8 bits . Using sizeof() we can find size of **data-types** or a **variable** also.

C Program to Find Size of Data Types

```
#include<stdio.h>
int main()
{
  printf("Size of short is %ld bytes\n",sizeof(short));
  printf("Size of int is %ld bytes\n",sizeof(int));
  printf("Size of long is %ld bytes\n",sizeof(long));

  printf("Size of float is %ld bytes\n",sizeof(float));
  printf("Size of double is %ld bytes\n",sizeof(double));
  printf("Size of long double is %ld bytes\n",sizeof(long double));

  printf("Size of char is %ld bytes\n",sizeof(char));
  printf("Size of void is %ld bytes\n",sizeof(void));
return 0;
}
```

```
Size of short is 2 bytes
Size of int is 4 bytes
Size of long is 8 bytes
Size of float is 4 bytes
Size of double is 8 bytes
Size of long double is 16 bytes
Size of char is 1 byte
Size of void is 1 byte
```

Floating-Point Types

Following table gives you detail about standard float-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. Following example will print storage space taken by a float type and its range values:

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

```
int main()
{
   printf("Storage size for float : %d \n", sizeof(float));
   printf("Minimum float positive value: %E\n", FLT_MIN );
   printf("Maximum float positive value: %E\n", FLT_MAX );
   printf("Precision value: %d\n", FLT_DIG );

   return 0;
}
```

When you compile and execute the above program it produces following result on Linux:

```
Storage size for float : 4
Minimum float positive value: 1.175494E-38
Maximum float positive value: 3.402823E+38
Precision value: 6
```

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

S.N.	Types and Description
1	**Function returns as void** There are various functions in C who do not return value or you can say they return void. A function with no return value has the return type as void. For example **void exit (int status);**
2	**Function arguments as void** There are various functions in C who do not accept any parameter. A function with no parameter can accept as a void. For example **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

The void type may not be understood to you at this point, so let us proceed and we will cover these concepts in upcoming chapters.

C - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive.

Variable Declaration in C

All variables must be declared before we use them in C program, although certain declarations can be made implicitly by content. A declaration specifies a type, and contains a list of one or more variables of that type as follows:

type variable_list;

Here, **type** must be a valid C data type including char, int, float, double, or any user defined data type etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid variable declarations along with their definition are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

You can initialize a variable at the time of declaration as follows:

```
int    i = 100;
```

Variable Initialization in C

Variables are initialized (assigned an value) with an equal sign followed by a constant expression. The general form of initialization is:

variable_name = value;

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

type variable_name = value;

Some examples are:

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

It is a good programming practice to initialize variables properly otherwise, sometime program would produce unexpected result. Try following example which makes use of various types of variables:

```
#include <stdio.h>

int main ()
{
  /* variable declaration: */
  int a, b;
  int c;
  float f;

  /* actual initialization */
  a = 10;
  b = 20;

  c = a + b;
  printf("value of c : %d \n", c);

  f = 70.0/3.0;
  printf("value of f : %f \n", f);

  return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
value of c : 30
value of f : 23.333334
```

Lvalues and Rvalues in C:

There are two kinds of expressions in C:

1. **lvalue :** An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
2. **rvalue :** An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement:

```
int g = 20;
```

But following is not a valid statement and would generate compile-time error:

```
10 = 20;
```

C - Constants and Literals

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are also enumeration constants as well.

The **constants** are treated just like regular variables except that their values cannot be modified after their definition.

Integer literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals:

```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */
```

Following are other examples of various type of Integer literals:

```
85         /* decimal */
0213       /* octal */
0x4b       /* hexadecimal */
30         /* int */
30u        /* unsigned int */
30l        /* long */
30ul       /* unsigned long */
```

C - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides following type of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators

- Assignment Operators
- Misc Operators

This tutorial will explain the arithmetic, relational, and logical, bitwise, assignment and other operators one by one.

Arithmetic Operators

Following table shows all the arithmetic operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

Following table shows all the arithmetic operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

Example

Try following example to understand all the arithmetic operators available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 21;
   int b = 10;
   int c ;

   c = a + b;
   printf("Line 1 - Value of c is %d\n", c );
   c = a - b;
   printf("Line 2 - Value of c is %d\n", c );
   c = a * b;
   printf("Line 3 - Value of c is %d\n", c );
   c = a / b;
   printf("Line 4 - Value of c is %d\n", c );
   c = a % b;
   printf("Line 5 - Value of c is %d\n", c );
   c = a++;
   printf("Line 6 - Value of c is %d\n", c );
   c = a--;
   printf("Line 7 - Value of c is %d\n", c );

}
```

When you compile and execute the above program it produces following result:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
```

```
Line 7 - Value of c is 22
```

Relational Operators

Following table shows all the relational operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

For example

Following table shows all the relational operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes	(A < B) is true.

	true.	
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Example

Try following example to understand all the relational operators available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 21;
   int b = 10;
   int c ;

   if( a == b )
   {
      printf("Line 1 - a is equal to b\n" );
   }
   else
   {
      printf("Line 1 - a is not equal to b\n" );
   }
   if ( a < b )
   {
      printf("Line 2 - a is less than b\n" );
   }
   else
   {
      printf("Line 2 - a is not less than b\n" );
```

```
   }
   if ( a > b )
   {
      printf("Line 3 - a is greater than b\n" );
   }
   else
   {
      printf("Line 3 - a is not greater than b\n" );
   }
   /* Lets change value of a and b */
   a = 5;
   b = 20;
   if ( a <= b )
   {
      printf("Line 4 - a is either less than or equal to  b\n" );
   }
   if ( b >= a )
   {
      printf("Line 5 - b is either greater than  or equal to b\n" );
   }
}
```

When you compile and execute the above program it produces following result:

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to  b
Line 5 - b is either greater than  or equal to b
```

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0 then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0 then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Example

Try following example to understand all the logical operators available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 5;
   int b = 20;
   int c ;

   if ( a && b )
   {
      printf("Line 1 - Condition is true\n" );
   }
   if ( a || b )
```

```
   {
      printf("Line 2 - Condition is true\n" );
   }
   /* lets change the value of  a and b */
   a = 0;
   b = 10;
   if ( a && b )
   {
      printf("Line 3 - Condition is true\n" );
   }
   else
   {
      printf("Line 3 - Condition is not true\n" );
   }
   if ( !(a && b) )
   {
      printf("Line 4 - Condition is true\n" );
   }
}
```

When you compile and execute the above program it produces following result:

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

Bitwise Operators

Bitwise operator works on bits and perform bit by bit operation. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; Now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13 then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13 then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either	(A | B) will give 61 which is

	operand.	0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Example

Try following example to understand all the bitwise operators available in C programming language:

```
#include <stdio.h>

main()
{

   unsigned int a = 60;          /* 60 = 0011 1100 */
   unsigned int b = 13;          /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;       /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;       /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;       /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );

   c = ~a;          /*-61 = 1100 0011 */
```

```
   printf("Line 4 - Value of c is %d\n", c );

   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

   c = a >> 2;     /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```

When you compile and execute the above program it produces following result:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

Assignment Operators

There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Example

Try following example to understand all the assignment operators available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 21;
   int c ;

   c =  a;
   printf("Line 1 - =  Operator Example, Value of c = %d\n", c );

   c +=  a;
   printf("Line 2 - += Operator Example, Value of c = %d\n", c );

   c -=  a;
   printf("Line 3 - -= Operator Example, Value of c = %d\n", c );

   c *=  a;
   printf("Line 4 - *= Operator Example, Value of c = %d\n", c );

   c /=  a;
   printf("Line 5 - /= Operator Example, Value of c = %d\n", c );

   c  = 200;
   c %=  a;
   printf("Line 6 - %= Operator Example, Value of c = %d\n", c );

   c <<=  2;
   printf("Line 7 - <<= Operator Example, Value of c = %d\n", c );

   c >>=  2;
   printf("Line 8 - >>= Operator Example, Value of c = %d\n", c );
```

```
   c &=  2;
   printf("Line 9 - &= Operator Example, Value of c = %d\n", c );

   c ^=  2;
   printf("Line 10 - ^= Operator Example, Value of c = %d\n", c );

   c |=  2;
   printf("Line 11 - |= Operator Example, Value of c = %d\n", c );

}
```

When you compile and execute the above program it produces following result:

```
Line 1 - =  Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - %= Operator Example, Value of c = 11
Line 7 - <<= Operator Example, Value of c = 44
Line 8 - >>= Operator Example, Value of c = 11
Line 9 - &= Operator Example, Value of c = 2
Line 10 - ^= Operator Example, Value of c = 0
Line 11 - |= Operator Example, Value of c = 2
```

Misc Operators ↦ sizeof & ternary

There are few other important operators including **sizeof** and **? :** supported by C Language.

Show Examples

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.

? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example

Try following example to understand the operator precedence available in C programming language:

```
#include <stdio.h>


main()
{
```

```
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;      // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );

   e = ((a + b) * c) / d;    // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);   // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;     //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
}
```

When you compile and execute the above program it produces following result:

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is  : 90
Value of (a + b) * (c / d) is  : 90
Value of a + (b * c) / d is  : 50
```

****