Chapter 3:- Introduction to C Programming

What is C?

C is a general-purpose, structured programming language. Its instructions consists of terms that resemble algebraic expression, augmented by certain English keywords such as if, else, for, do and while, etc. C contains additional features that allow it to be used at a lower level, thus bridging the gap between machine language and the more conventional high level language. This flexibility allows C to be used for system programming (e.g. for writing operating systems as well as for applications programming such as for writing a program to solve mathematical equation or for writing a program to bill customers). It also resembles other high level structure programming language such as Pascal and Fortran.

History of C

C was an offspring of the 'Basic Combined Programming Language' (BCPL) called B, developed in 1960s at Cambridge University. B language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972. The new language was named C. Since it was developed along with the UNIX operating system, it is strongly associated with UNIX. This operating system was developed at Bell Laboratories and was coded almost entirely in C.
C was used mainly in academic environments for many years, but eventually with the release of C compiler for commercial use and the increasing popularity of UNIX, it began to gain widespread support among compiler professionals. Today, C is running under a number of operating systems including Ms-DOS. C was now standardized by American National Standard Institute. Such type of C was named ANSI C.

Features of C

- It is a robust language with rich set of built-in functions and operators that can be used to write any complex program.
- The C compiler combines the capabilities of an assembly language with features of a high-level language.
- Programs Written in C are efficient and fast. This is due to its variety of data type and powerful operators.
- It is many time faster than BASIC.
- C is highly portable this means that programs once written can be run on another machines with little or no modification.
- Another important feature of C program, is its ability to extend itself.
- A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.
- C language is the most widely used language in operating systems and embedded system development today.

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

Basic Structure of C programs:
A C program may contain one or more sections shown in fig:

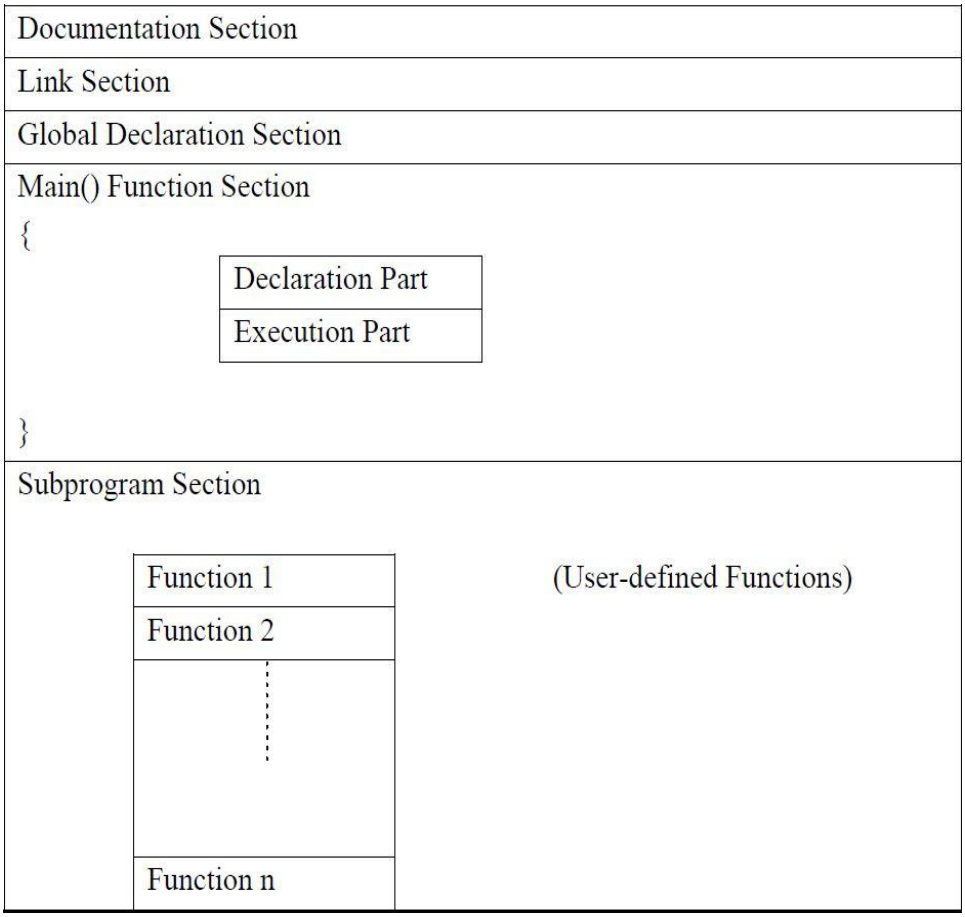


Fig: Basic Structure of a C program

Documentation Section
This section consists of comment lines which include the name of programmer, the author and other details like time and date of writing the program. Documentation section helps anyone to get an overview of the program.

Link Section
The link section consists of the header files of the functions that are used in the program. It provides instructions to the compiler to link functions from the system library.

Definition Section
All the symbolic constants are written in definition section. Macros are known as symbolic constants.

Global Declaration Section
The global variables that can be used anywhere in the program are declared in global declaration section. This section also declares the user defined functions.

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

main() Function Section

It is necessary have one main() function section in every C program. This section contains two parts, declaration and executable part. The declaration part declares all the variables that are used in executable part. These two parts must be written in between the opening and closing braces. Each statement in the declaration and executable part must end with a semicolon (;). The execution of program starts at opening braces and ends at closing braces.

Subprogram Section

The subprogram section contains all the user defined functions that are used to perform a specific task. These user defined functions are called in the main() function.

3.1 Character Set, Keywords and Data types

Character Set

1. Alphabets

- Uppercase letters A-Z
- Lowercase letters a-z

2. Digits

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

3. Specials Character

~ → tilde	% → percent sign	| → vertical bar
@ → at symbol	+ → plus sign	< → less than
_ → underscore	- → minus sign	> → greater than
^ → caret	# → number sign	= → equal to
& → ampersand	$ → dollar sign	/ → slash
) → right parenthesis	(→ left parenthesis	* → asterisk
\ → back slash	′ → apostrophe	, → comma
: → colon	. → Dot operator	] → right bracket
[→ left bracket	" → quotation mark	; → semicolon
! → Exclamation mark	? → Question mark	{ → left flower brace
} → right flower brace		

4. White Characters

\b → blank space	\t → horizontal tab
\r → carriage return	\f → form feed
\\ → Back slash	\’ → Single quote
\" → Double quote	\v → vertical tab
\? → Question mark	\0 → Null
\a → Alarm (bell)	\n → new line

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

Keywords

There are certain reserved words, called keywords that have standard, predefined meanings in C. These keywords can be used only for their intended purpose; they cannot be used as programmer-defined identifiers. The standard keywords are:

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

The keywords are all lowercase. Since upper and lowercase characters are not equivalent, it is possible to utilize an uppercase keyword as an identifier. Normally, however, this is not done, as it is considered a poor programming practice.

Identifiers:

Identifiers are names that are given to various program elements, such as variables, functions and arrays. Identifiers consisted of letters and digits, in any order, except that first character must be a letter. Both upper and lower case letters are permitted, though common usage favors the use of lowercase letters for most type of identifiers. Upper and lowercase letters are not interchangeable (i.e. an uppercase letter is not equivalent to the corresponding lowercase letters). The underscore (_) can also be included, and considered to be a letter. An underscore is often used in middle of an identifier. An identifier may also begin with an underscore.

Rules for Identifier:

- First character must be an alphabet (or Underscore).
- Must consist of only letters, digits or underscore.
- Only first 31 characters are significant.
- Cannot use a keyword.
- Must not contain white space.

Data Types

The data type in **C** defines the amount of storage allocated to variables, the values that they can accept, and the operation that can be performed on those variables.

C is rich in data types. The verity of data type allow the programmer to select appropriate data type to satisfy the need of application as well as the needs of different machine.

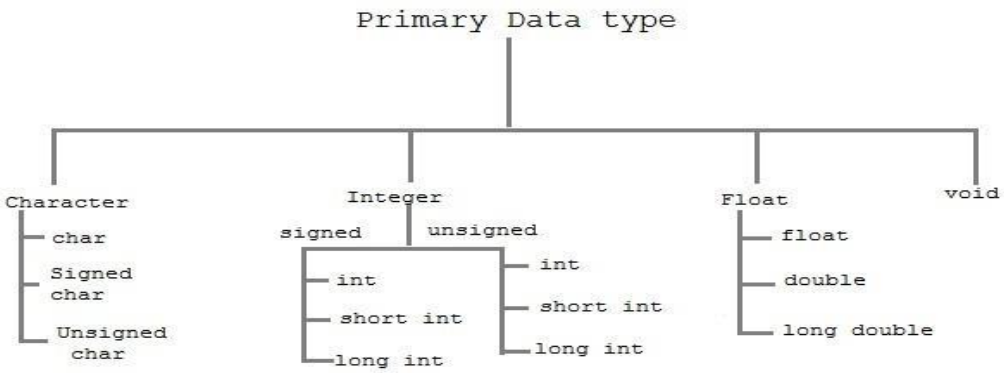
There are following type of data types supported by c programming

- Primary Data Type
- Derived Data Type
- User Defined Data Type

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

Primary (Fundamental) Data type

All C compiler support following fundamental data type



Integer Type

Integers are the whole numbers, i.e. non-fractional numbers. They are defined in c by keyword int. Generally integer require 16 bit of storage. Integers can be of following types:

Type	Represented by	Conversion Character	Memory occupied
Signed Integer	signed int or int	%d	2 byte
Unsigned Integer	unsigned int or unsigned	%u	2 byte
Signed Short Integer	signed short int or short int or short	%d or %i	2 byte
Unsigned Short Integer	unsigned short int or unsigned short	%u	2 bute
Signed Long Integer	signed long int or long int	%ld	4 byte
Unsigned Long Integer	unsigned long int or unsigned long	%lu	4 byte

Floating Point Types

Floating types are fractional numbers (i.e. real numbers). They are defined in c by keyword float. Generally floating numbers reserve 32 bits of storage.

Type	Represented by	Conversion Character	Memory Occupied
Floating Point	Float	%f	4 byte
Double	double	%lf	8 byte
Long Double	long double	%Lf	10 byte

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

Void Type
The void type has no value. It is usually used to specify type of function when it does not return any value to calling function.

Character Type
A single character can be defined as a character type data. Characters are stored in 8 bits. It is represented by a keyword char. Its conversion character is %c.

User Defined Data types
The user defined data types enable a program to invent his own data types and define what values it can take on. Thus these data types can help a programmer to reducing programming errors.
C supports 2 types of user defined data types.

- typedef (type definition)
- enum (enumerated data type)

Eg:- typedef int integer;
Here, integer symbolizes int data type. Now we can declare int variable as a integer instead of int like:-
integer num; which is equivalent to int num;

Derived Data Type
Data types that are derived from the built-in data types are known as derived data types. The various derived data types provided by C are arrays, pointers and structures. For example :

struct st1{

int a;

float b;

char c;} Here 'struct' is a derived data types 'structure'. It consists of integer, float and character variables. Structure, unions and enumerations are the derived data types used in C.

Data Types Summary

Data Type	Category	Range	Memory Space (byte)	Format Specifier (Conversion Factor)
integer	short signed short unsigned	-32,768 to 32,768 0 to 65536 -32,768 to 32,768	2 2 2	%d or %i %u %d

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

	signed unsigned long signed long unsigned hexadecimal unsigned octal	0 to 65535 -2,147,483,648 to 2,147,483,648 0 to 4,294,967,295	2 4 4	%u %ld %lu %x %u
float	Float	1.2x10^(-38) to +3.4x10^(+38)	4	%f
double	double long double	1.7x10(^-308) to 3.4x10^(+308) 3.4x10^(-4932) to 1.1x10^(+4932)	8 10	%lf %Lf
char	signed char unsigned char	-128 to +127 0 to 255	1 1	%c %c

3.2 Preprocessor and Directives

The C preprocessor is a collection of special statements, called directives that are executed at the beginning of compilation process. Preprocessors directives usually appear at the beginning of a program. A preprocessor directive may appear anywhere within a program. Preprocessor directives follow special syntax rules that are different from the normal C syntax. They all begin with the symbol # in column one and do not require a semicolon at the end. We have already used the directives #define and #include to a limited extent. A set of commonly used preprocessor directives and their functions are listed below:

Directives	**Functions**
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Species the files to be included
#ifdef	Test for a macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether a macro is not defined
#if	Tests a compile-time condition
#else	Specifies alternatives when #if test fails.

Symbolic Constants

A symbolic constant is name that substitute for a sequence of character that cannot be changed. The character may represent a numeric constant, a character constant, or a string. When the program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence. They are usually defined at the beginning of the program. The symbolic constants may then appear later in the program in place of the numeric constants, character constants, etc., that the symbolic constants represent.
For example:

C program consists of the following symbolic constant definitions.

```
#define PI 3.141593
#define TRUE 1
#define FALSE 0
```


By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

#define PI 3.141593 defines a symbolic constant PI whose value is 3.141593. When the program is preprocessed, all occurrences of the symbolic constant PI are replaced with the replacement text 3.141593.

C tokens:

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual units in a C program are known as C tokens.

C tokens are of six types. They are:
1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Strings (eg: “total”, “hello”),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /,-,*)

3.3 Constants/Literals and Variables

A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5, "C programming is easy", etc.

1. Integer constants
An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

For example:
Decimal constants: 0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc
In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

2. Floating-point constants
A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:
-2.0
0.0000234
-0.22E-5
Note: E-5 = 10-5

3. Character constants

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

A character constant is a constant which uses single quotation around characters. For example: 'a', 'l', 'm', 'F'

4. String Constant

A string constant is a sequence of characters enclosed in double quotes. It may contain letters, numbers, special characters or blank spaces. However it does not have equivalent ASCII values. Examples:- "Hello", "2008", "5+3" etc.

Variables:

A variable is an identifier that is used to represent some specified type of information within a designated portion of the program. In its simplest form, a variable is an identifier that is used to represent a single data item, i.e., a numerical quantity or a character constant. The data item must be assigned to the variable at some point in the program. A given variable can be assigned different data items at various places within the program. Thus, the information represented by the variable can change during the execution of the program. However, the data type associated with the variable are not change.

Eample: *int num*; Here num is an integer type variable.
Float num; Here num is an floating type variable.
The declaration of variables mainly has two significances:

- It gives name of variables and its type.
- It allocates appropriate memory space according to its data types.

Rules for naming variables:

- First character must be an alphabet (or Underscore).
- Must consist of only letters, digits or underscore.
- Only first 31 characters are significant.
- Cannot use a keyword.
  Must not contain white space.

3.4 Operators and Statements

C supports a rich set if build in operators. An operator is a symbol that tells the computer to perform mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.
C operators can be classified into a number of categories. They include:

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and Decrement operators
- Conditional operators

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

- Bitwise operators
- Special operators

Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning of Operator	Example

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

&&	Logial AND. True only if all operands are true .	If c = 5 and d = 2 then, expression ((c==5) &&(d>5)) equals to 0.
||	Logical OR. True only if either one operand is true.	If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1.
!	Logical NOT. True only if the operand is 0.	If c = 5 then, expression ! (c == 5) equals to 0.

Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Increment and decrement operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Both the increment and decrement operators can either precede (prefix) or follow (postfix) the operand. For example −

```
x = x+1;
```

can be written as

```
++x; // prefix form
```

or as −

```
x++; // postfix form
```

When an increment or decrement is used as part of an expression, there is an important difference in prefix and postfix forms. If you are using prefix form then increment or decrement will be done before rest of the expression, and if you are using postfix form, then increment or decrement will be done after the complete expression is evaluated.

Below table will explain the difference between pre/post increment and decrement operators in C programming language.

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

Operator	Operator/Description
Pre increment operator (++i)	value of i is incremented before assigning it to the variable i
Post increment operator (i++)	value of i is incremented after assigning it to the variable i
Pre decrement operator (–i)	value of i is decremented before assigning it to the variable i
Post decrement operator (i–)	value of i is decremented after assigning it to variable i

Bitwise Operators

In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level. To perform bit-level operations in C programming, bitwise operators are used.

1. Bitwise AND operator (&)

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bit Operation of 12 and 25
00001100
& 00011001

_____
00001000 = 8 (In decimal)

2. Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise OR Operation of 12 and 25
00001100
| 00011001

_____
00011101 = 29 (In decimal)

3. Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise XOR Operation of 12 and 25

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

00001100
| 00011001

_____
00010101 = 21 (In decimal)

4. Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)

5. Left Shift Operator

Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by <<.

212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 =11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)

Conditional (Ternary) Operator (? :)

The operator?: is known as conditional operator. Simple conditional operations can be carried out with conditional operator. An expression that make use of the conditional operator is called a conditional expression. Such an expression can be written in place of traditional if else statement.

A conditional expression is written in the form:

expression1? expression2: expression3

When evaluating a conditional expression, expression1 is evaluated first. If expression1 is true, the value of expression2 is the value of conditional expression. If expression1 is false, the value of expression3 is the value of conditional expression. For example:

```
a = 10 ;
b = 15 ;

x = (a > b)? a: b ;
```

In this example, x will be assigned the value of b. This can be achieved by using the if else statement as follows:

```
if (a<b)
        x = a;
else
        x = b;
```


By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

Special Operators

C supports some special operators such as comma operator, size of operator, pointer operator (* and &) and member selection operators.

- **Comma Operator:**
  The comma operator can be used to link the related expression together. A comma linked list of expression are evaluated left to right and the value of right-most expression is the value of combined expression. For example,
  Value = (x = 10, y = 5, x + y),
  Here, 10 is assigned to x and 5 is assigned to y and so expression x+y is evaluated as (10+5) i.e. 15.
- **Size of Operator:**
  The size of operator is used with an operand to return the number of bytes it occupies. It is a compile time operand. The operand may be a variable, a constant or a data type qualifier. The associativity of size of right to left. For e.g.
  Suppose that i is an integer variable, x is a floating-point variable, d is double-precision variable and c is character type variable.
  The statements:
  printf ("integer : %d \n", sizeof(i)) ;
  printf("float : %d \n", sizeof(x)) ;
  printf("double : %d \n", sizeof(d)) ;
  printf ("character : %d \n", sizeof(c)) ;
  might generate the following output : integer :2 float :4 double :8

Operator precedence and associativity

The data type and the value of an expression depends on the data types of the operands and the order of evaluation of operators which is determined by the precedence and associativity of operators. Let us first consider the order of evaluation. When expressions contain more than one operator, the order in which the operators are evaluated depends on their precedence levels. A higher precedence operator is evaluated before a lower precedence operator. If the precedence levels of operators are the same, then the order of evaluation depends on their associativity (or, grouping). In Chapter we briefly discussed the precedence and associativity of arithmetic operators.

Summary of operators their precedence and associativity.

Operators	**Description**	**Associativity**	**Precedence Rank**
() []	Function call Array element reference	Left -> Right	1
+ - ++ -- ! ~ *	Unary Plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference	Right -> Left	2

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

& sizeof	Address Sizeof operator	Right -> Left	2
* / %	Multiplication Division Modulus	Left -> Right	3
+ -	Addition Subtraction	Left -> Right	4
<<	Leftshift		
>>	Rightshift		
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left -> Right	5
== !=	Equality Inequality	Left -> Right	6
&	Bitwise AND		7
^	Bitwise XOR		8
|	Bitwise OR		9
&&	Logical AND		10
||	Logical OR		11
?:	Conditional operator		12
= += -= *= /= %=	Assignment operator	Right -> Left	13
,	Comma operator	Left -> Right	14

Type Conversion

When variables and constants of different types are combined in an expression then they are converted to same data type. The process of converting one predefined type into another is called type conversion.

Type conversion in c can be classified into the following two types:

1. Implicit Type Conversion

When the type conversion is performed automatically by the compiler without programmer's intervention, such type of conversion is known as implicit type conversion or type promotion**.** The compiler converts all operands into the data type of the largest operand.

Example of Type Implicit Conversion:

```
#include<stdio.h>
int main()
{
   int x = 10;    // integer x
```


By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad

```
    char y = 'a';  // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```
Output:
x = 107, z = 108.000000

2. Explicit Type Conversion The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion. The explicit type conversion is also known as type casting**.**
Type casting in c is done in the following form:
(data type)expression;
where, *data type* is any valid c data type, and *expression* may be constant, variable or expression.

Example of Type Explicit Conversion :
```
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```
Output:
sum = 2

Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.

By: Deepak Kr. Singh, Pradip Khanal
Anku Jaiswal, Prakash Chandra Prasad