Chapte 1

Introduction to C

1.1 OVERVIEW OF C

1.1.1 Introduction

C is a high level Computer Programming Language. It is the most widely used language for systems programming work like design of Operating Systems, Compilers, Editors, Assemblers, various software tools, software utilities, etc. This language can also be used for small business data processing applications.

Originally, the language **B** was developed by Ken Thomson in 1970 for the **UNIX** operating system on DEC series PDP-7 computer. For many years, the description of C was the reference manual in the first edition of the **C programming language**. In 1983, the American National Standard Institute (ANSI) established a committee for complete definition of C and the same was completed in 1988. Now C is called as **ANSI C**.

1.1.2 Importance of C Language

There are several important features of C language that makes it as a popular one. Few of them are described below:

- C is **highly portable**. This means that it is independent of any particular machine. With small changes, it is easy to write a C program that can run on any platform like UNIX, DOS, Windows, etc. This portability is achieved by having a collection of standard **header files**. These header files combined with

the standard library routines (I/O functions in particular), one can access input and output devices.

- C is **fast**. There are many reasons for this. We shall give few of them:
  C has many standard operators like ++, --, %, +=, -=, *=, /=, etc. which correspond to direct hardware operations.
- C uses **pointers** through which the data can be stored or retrieved in memory. In other words, pointer variables can hold memory addresses and access the data.
- C variables can be initialized during declarations itself there by **execution time** is reduced.
- C is **compact**. C combines some complicated operations or statements into special operators. For example, the assignment statement `a = a + b;` can be written as `a += b`, where += is an operator. Similarly, `i = i + 1;` can be written as `i++`.
- C allows both **low-level** and **high-level** programming. It simply means that the hardware memory, registers, etc. can be accessed and manipulated in C programs. For example, bit manipulations, register variable declarations, octal and hexadecimal data representation, etc. are all possible in C programs. In other words the data types and control structures provided by C are supported directly by most computers.
- C offers only straight forward, **single thread** control flow: tests, loops, grouping, and subprograms, but not multiprogramming, parallel operations, synchronization, or co-routines.

Disadvantages

- Though pointers offer direct access of memory and efficient character string manipulations, sometimes it may be confusing and dangerous.
- Some symbols in C are ambiguous – for example, & is an address operator, && is a logical operator and & is a bit-wise operator.

1.1.3 Basic Structure of a C Program

This section gives an outline of a **typical C program**. Some of the sections may be optional or all sections need not be there in all programs.

The basic building block of C programs is the **function**. A function is a separate block of code that performs a specific task (it is also called as a **module** or a **subprogram**). Each function will contain statements that carry out the task of the function. The Figure 1.1 shows the basic structure of a C program.

When variables are declared outside the function(s), it is called as **global variables** and if the variables are declared inside the function(s), then they are called as **local variables**. The global variables are accessible by all the functions. However, the local variables are visible only with in the scope of that particular function.

```
/* preprocessor directives           */
/* function prototype definitions    */

/* global variable declaration       */

/* function name or function header  */
/* main() function                   */
/* body of the function              */

/* user defined function(s)          */
/* body of the function              */

#include <stdio.h>
#define MAX 10
int x; /* global variable */

void main()
{
     variable declarations; /* local variables */
     statement-1;
     statement-2;
     .....
     statement-n;
}

int func1(parameters)       /* function header */
{
     variable declarations; /* local variables */
     statement-1;
     statement-2;
     .....
     statement-n;
}
```

Fig. 1.1 Structure of a typical C program.

The `<stdio.h>` is a standard header file that appears as a first line in most of the C programs. This is known as the **standard input/output header file**, it copies an external header file into the source file at that location.

Function **prototype** statements provide information that describes each function (except `main()`) defined in the program.

Generally every C program should have a predefined function called `main()`. In fact, the execution of a C program starts from the first statement in `main()`. One can place functions (with out `main()`) in a separate file and compile the same. This is called as **separate compilation**.

Every function will have one or more declarations and statements. A program executes statements in the body of the function in the order in which it appears from top

to bottom. However, when control statements or looping is used, then the flow of execution may get altered.

There is no pre-defined order of placing the functions but Figure 1.1 is one possible way that is used by many C programmers.

1.1.4 Programming Style

Every programming language has a particular style which indicates the way in which the statements are written and placed in a file. In C language we can place the statements, declarations, etc. any where in the file, but proper indentation is very important. The C programs should have appropriate **comments**. The comments in C has a syntax:

```
/* this is a comment line */
```

In UNIX or TURBO-C or Borland C compiler, all C programs are written in lowercase letters only. Uppercase letters may be used for defining constants, variables, etc. In a single line, multiple statements can appear. For example,

```
x = 1;
y = 2;
z = 3;
```

can also be written as,

```
x = 1; y = 2; z = 3;
```

Looking at another example,

```
if (x <= 10) z = x + 1;
```

Writing comments is one of the prime requirements for a good programming style. This is mainly to increase the readability of the program.

1.1.5 Sample C Programs

Though it is too early to see any example C program, it would enable the reader to get some idea of a C program. The aim of this example is to print **"Hello!"** on the screen. The Program 1.1 does this.

Program 1.1
Printing Hello

```
#include <stdio.h>
void main()
{
    printf ("Hello\n");

}
```

- `stdio.h:` Standard header file.
- `void main():` main function and receives no arguments.
- `printf():` calls a built-in function `printf()` to print all the characters between " ". The '\n' is used for new line. The braces { } are used for the body of the function.

Program 1.2
Adding two data elements

```
/* Program to add two data elements */
/* Author : S. Nandagopalan Date : 19/09/2001 */
#include <stdio.h>
void main()
{
     int accno; /* account number - line - 1 */
     float deposit, balance;      /* line - 2  */
     accno = 1050;                /* line - 3  */
     balance = 1700.50;           /* line - 4  */
     deposit = 226.50;            /* line - 5  */
     balance = balance + deposit;     /* update balance */
                                      /* line - 6 */
     printf ("Account No = %d\n", accno); /*line - 7*/
     /* line - 8 */
     printf ("Current balance = %6.2f\n", balance);
} /* end of main,  line - 9*/
```

line –1 and line – 2: These two lines are simple *integer* and *float* variable declarations.

lines –3, 4 and 5: These lines are assignment statements which assign the initial values for the account number (`accno`), `balance` and `deposit`.

line – 6: This line is also an assignment statement, that adds `balance` and `deposit` i.e. 1927.00. This updated value will in turn be assigned to `balance`.

lines - 7 and 8: The `result` and the account number will be printed on the monitor. Line – 7 prints the account number in *integer* format, whereas line – 8 prints in *floating* point format.

line – 9: The brace } indicates the end of `main` program.

Program 1.3
User defined function - demo

```
#include <stdio.h>
float transaction (float, float); /* line -1 */
void main()
{
     int accno;
     float deposit, balance;
     accno = 1050;
     balance = 1700.50;
     deposit = 226.50;
     balance = transaction(balance, deposit);/* line - 2 */
     printf ("Account No = %d\n", accno);
     printf ("Current balance = %6.2f\n", balance);
}
float transaction (float b, float d) /* line - 3 */
{
     float temp;
     temp = b + d;
     return temp; /* line - 4 */
}
```

line – 1: is a prototype declaration for the user defined function called `transaction()`.

line – 2: is the calling sequence of the function. The values sent to the function are `balance` and `deposit`.

1.2 Constants, Variables and Data Types

1.2.1 Introduction

This chapter introduces the meaning of constants, variables and various other data types that are the basic building blocks of C statements. All these statements put together form a program.

Every language is based upon an alphabet of character. The English alphabet, for example, consists of 26 letters A to Z. Similarly, the decimal system uses the digits 0, 1, 2, ...,9. In the same way, C language also uses a set of characters. Today most of the Personal Computers follow **ASCII** (American Standard Code for Information Interchange) character set. The program statements must follow the syntax of the C language. The **syntax** is a set of rules which determines whether the sentence is well formed or not.

1.2.2 Character Set

C language characters are mainly from the ASCII character set and grouped in to four categories:

- letters
- digits
- special characters
- white spaces

Letters

The C letters are

```
A, B, C,.....,Z
a, b, c,.....,z
```

Digits

The digits of C language are

```
0, 1, 2,.....,9
```

Special Characters

Sl. No.	Character	Meaning
1	,	comma
2	.	period
3	;	semi colon
4	:	colon
5	?	question mark
6	'	apostrophe
7	"	quote
8	!	exclamation
9	|	vertical bar
10	/	slash
11	\	back slash
12	~	tilde
13	_	underscore
14	$	dollar
15	#	hash
16	&	ampersand
17	^	caret
18	*	asterik
19	-	minus
20	%	percent
21	+	plus
22	<	less than

23	>	greater than
24	(	left parenthesis
25	)	right parenthesis
26	[	left bracket
27	]	right bracket
28	{	left brace
29	}	right brace

White Space

- Blank space
- Horizontal tab
- Carriage Return
- New Line
- Form feed

1.2.3 C Tokens

In a C source file, each word, and punctuation mark are called as tokens. There are six classes of tokens namely,

1. Identifiers
2. Constants
3. Keywords
4. String Literals
5. Operators
6. Other separators

White space characters are ignored except as they separate tokens. Some white space is required to separate, otherwise adjacent identifiers, keywords and constants. We shall try to understand each of these tokens in detail.

Identifier

An **identifier** is a sequence of letters and digits. The first character must be a letter (the underscore character _ is also taken as a letter). For example,

```
account          MAX
deposit          Max
n                bubble_sort
file1
```

C language is case sensitive, i.e. uppercase and lowercase characters are treated differently. In the above example `MAX` and `Max` are different identifiers. Following are some invalid identifiers:

```
9amount          -    first letter can not be a digit
char             -    key word
```

```
bubble-sort      -     - is an operator
auto             -      key word
selection sort   -     space not allowed
```

Keywords

Every programming language uses its alphabet to form words or symbols that make up the vocabulary of the language. Many of the special characters have a well-defined meaning in certain context and termed as a **keyword**. In C there are some **reserved words** that have specific meaning and can not be used as variable names.
The advantages of reserved words are:

- They help to make the program more readable.
- They permit the compiler to speed up its compilation process.
- They facilitate error recovery.

However, when the number of reserved words grows it is difficult to remember all of them. The following identifiers are reserved for use as keyword:

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

Some of the extended keywords are listed below:

far	*fortran*	*huge*	*interrupt*
near	*pascal*	*asm*	

Constants

Following are the types of constants used in C language:

- Integer constant
- *float*ing point or real constant
- character and string constant
- enumeration constant

Integer constant

There are three types of *int*eger constants, depending upon its form, value and suffix – namely **decimal, octal** and **hexadecimal.**

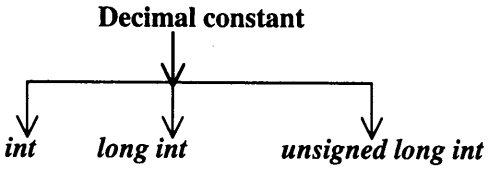
Decimal constants are a sequence of digits 0, 1, 2,.....,9 preceded by an unary operator + or -. Below are few examples,

56 -4 30715 +99

You can not add any other non-digit characters like, or $ or Rs. or :

7,525	(, is not permitted)
$10,000	($ and, not permitted)
Rs. 134.50	(Rs not permitted)

An *int*eger constant may be suffixed by the letter `u` or `U` to specify that it is unsigned. When a constant is suffixed by `l` or `L` then it represents *long int*eger.



For example,

i) 42756u	or	42756U	(unsigned *int*eger)
ii) 9786251ul	or	9786251UL	(unsigned *long int*)
iii) 17526l	or	17526L	(*long int*)

(b) Floating point or Real Constant

A **floating point** constant consists of an *int*eger part, a decimal point and a fractional part. For example, in 175.26, the 175 is the *int*eger part and 26 is the fractional part. More examples are shown below:

3.141590.0001	75.
.001	1.2

The **floating point** numbers are useful when there is a requirement for more accuracy like scientific calculations and financial or business applications.

The keyword *double* is same as *float* except that the memory allocated is twice as that of the *float*. The *float* type occupies 4 bytes and double takes 8 bytes. A very important point which the reader must remember is that, at execution time the value of a character constant is the numeric value of that character. For example,

'a' is stored as ASCII value 97 decimal.

Therefore, you can perform character arithmetic like the one shown below:

```
'B' + 1 yields 'C'
```

This means that the ASCII value of 'B' is added with 1 which gives away the result 'C'.

Escape sequence characters

There are certain special **escape sequence** character constants which have specific meaning. The following Table 1.1 gives all the escape characters of C.

String Constant

A **string** is a collection of one or more ASCII characters enclosed with in a pair of double quotes. For example,

"Bangalore"
"Hi!"
"Pin-code 560 004"
"T"

Table 1.1 Escape sequence characters

newline	\n	backslash	\\
horizontal tab	\t	question mark	\?
vertical tab	\v	single quote	\'
back space	\b	double quote	\"
carriage return	\r	octal number	\000
form feed	\f	hex number	\xhh
bell	\a		

Note
\000 - specifies an octal digit to represent the desired character.
\xhh - x followed by hexadecimal digits to specify the desired character.
\0 - represents ASCII NULL. This is used to terminate a character string.

Enumeration constant

Enumerations are unique types with values ranging over a set of name constants called **Enumerators**. Identifiers declared as enumerators are constants of type*int*. You will see more details about enumerations in later chapters.

1.3 Data Types

There are only basic or primitive data types in C.

Sl. No.	Data Type	Description	Size
1	char	character	1 byte
2	int	integer	2 bytes
3	float	single precision floating point	4 bytes
4	double	double precision floating point	8 bytes

In addition to the above data types, C also provides short and *long* types.

Integer Data Type

The *int* data type will be the natural size of the host machine and *short* is often 16 bits (2 bytes), *long* is 32 bits (4 bytes) and *int* is either 16 or 32 bits depending upon the machine.

Assignment and Initlization

Once the variable is declared, it is ready to take on values depending upon the type of the variable. The syntax of an assignment statement is:

```
var_name = constant or var_name;
```

Here, the = is used as an assignment operator. Below are few examples:

```
int n;
n = 10; /* asssignment statement */
char c;
c = 'A';
```

When a value is specified for a variable during declaration, then its is called as **initlization**. For example,

```
int n = 10;
char c = 'A';
```

The advantage of initlization is that the value of the variable n or c is ready at the time of compilation itself. Of course, it is possible to assign other values to these variables later in the program.

Multiple assignment operation

A single value can be assigned to multiple variables with in a single assignment statement. See the below example statements:

```
a = b = c = 0;
n = MAX = 80;
```

Declaring Symbolic Constants

Symbolic constants are variables that are same as the associated constants and are declared as,.

```
#define TRUE   1
#define FALSE  0
```

The symbolic constants are normally handled by the preprocessor of the compiler. C **preprocessing** is a process that occurs before actual compilation begins. The preprocessor is part of the main compiler. It is the first step of the compilation process and the output of this will be given to the later steps of the compilation process.

The preprocessor directives start with a `#define` folllowed by an identifier and then followed by a constant as shown above. During the preprocessing, the compiler replaces all occurrences of these identifiers with the values specified. Unlike a variable, the value of the symbolic constant can not be changed.

```
#define n 10
    .....
n = 20; /* illegal */
```

Declaring a variable Constant

ANSI standard has introduced a new way of declaring a variable which could act like a constant by using const declaration (see below).

```
const int n = 10;
const float pi = 3.1415;
```

The value of n and pi can not be changed during program execution. This kind of declaration helps the program to execute faster than a preprocessor way of declaration (i.e. `#define`).

User defined type declaration

Sometimes it is more readable and meaningful to define our own data types. For example, to add a new data type `UCHAR; unsigned char` the following declaration can be used:

```
typedef unsigned char UCHAR;
```

Later, you can make declarations as,

```
UCHAR ch1, ch2;
```

The syntax for the used defined data type is:

```
typedef basic_data_type user_defined_name;
```

Take a look at another example,

```
typedef char String[80];
String message; /* message is of type String */
```

Enumerated data type declaration

When you want to declare variables with an associated set of constant values, C provides the best method called as enumeration (enum) data type. The syntax and example is shown below:

```
enum tag { enumeration list };
enum boolean {NO, YES};
```

The first name in enum has a value 0, thenext 1, and so on, unless explicit values are specified.

```
enum days_of_week {MON, TUE, WED, THU, FRI, SAT, SUN};

enum months {JAN = 1,FEB,MAR,APR,MAY,JUN,
                    JUL,AUG,SEP,OCT,NOV,DEC};
/* JAN = 1, FEB = 2,MAR = 3, etc. */
```

Later variables can be declared of enum type, as shown below:

```
boolean flag;  /* flag can take values YES or NO */
```

Example Program 1.5
Integer constant - demo.

```
#include <stdio.h>
void main()
{
     int dec = 16; /* decimal */
     int oct = 020; /* octal */
     int hex = 0x10; /* hexadecimal */
     printf("\n Answer in decimal = %d %d %d",
                    dec,oct,hex);
     printf("\n Answer in octal = %o %o %o",
                    dec,oct,hex);
     printf("\n Answer in hexadecimal = %x %x %x",
                    dec,oct,hex);
}
```

Sample Run

```
Answer in decimal = 16 16 16
Answer in octal = 20 20 20
Answer in hexadecimal = 10 10 10
```

The `printf()` function prints the values of variables on the console as per the conversion modifiers (like `%d`, `%o`, `%x`, etc.) given as the first argument. We shall study moreabout this function in the next chapter.

Program 1.6
long integer *constants - demo*

```
#include <stdio.h>
void main()
{
     long ldec, loct, lhex;
     ldec = 16L;
     loct = 020L;
     lhex = 0x10L;
     printf("\n Long decimal = %ld", ldec);
     printf("\n Long octal = %lo", loct);
     printf("\n Long hex = %lx", lhex);
}
```

Sample Run

```
Long decimal = 16
Long octal = 20
```

```
Long hex = 10
```

For *long int* the coversion modifier to be used is `ld` or `lo` or `lx`, etc.

Example Program 1.7
Character declaration and arithmetic.

```
#include <stdio.h>
void main()
{
     char c = 'A';
     char x;
     x = c + 1; /* ASCII value of 'A' + 1 */
     printf("ASCII code of A = %d\n", c);
     printf("x = %c\n", x);
}
```

Sample Rub

```
ASCII code of A = 65
x = B
```

1.5 MANAGING INPUT AND OUTPUT OPERATIONS

1.5.1 Introduction

The input and output operations are managed through a set of I/O functions. For any programming activity, the data has to be read from some input device and the results are to be sent to the output device. In most of the cases, the default input device is a **keyboard** and the default output device is a **monitor**. In general, C operates with input and output in terms of **data streams**. A **stream** is a flow of data from keyboard to the system or from system to monitor.

To handle the I/O activity every computer language has reserved commands (not functions). However, in C, I/O is done through a set of functions like `getchar()`, `putchar()`, `gets()`, `puts()`, etc. These set of I/O functions are stored in include files called as `<stdio.h>`. You would have noticed in the programs of earlier chapters that the purpose of the `#include <stdio.h>` statement is to inform the compiler about the prototypes of I/O functions.

1.5.2 Unformatted single Character Reading and Printing

getchar() and putchar()

The function `getchar()` reads a single character from the keyboard and `putchar()` writes a single character to the console.

You can read a character from the keyboard and assign it to the variable c with the following statement

```
c = getchar();
```

We shall see in the following lines, how `getchar()` works. When the function `getchar()` is executed, it waits for the user to enter a character followed by enter key (Carriage Return <CR>). The character entered is assigned to the variable c. However, when the user enters more than on character and then press the *Enter key* – what happens?

Well, most of the IBM keyboards are **buffered**, that is, whenever a key is pressed, the character goes to a buffer (memory) in the keyboard. When more characters are pressed, all of these characters go to the buffer. The below Figure 1.3 illustrates the keyboard contents after the user has pressed `abcdef<CR>`

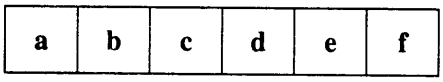


Fig. 1.3 Keyboard buffer

When you press <CR>, the first character from the buffer is assigned to the variable c. The remaining characters in the buffer will stay in the buffer itself. It is possible to clear the keyboard buffer by using `fflush()` function. The character read using `getchar()` can be displayed on the screen with `putchar()` function.

Program 1.8
getchar from keyboard

```
#include <stdio.h>
void main()
{
    char c;
    printf("Enter a character: ");
    c = getchar();
    putchar(c);
}
```

Sample Run

```
Enter a character: a
a
```

1.5.3 Formatted I/O - scanf() and printf()

The `scanf()` function reads characters from the keyboard, interprets them according to the format specification given in the first argument and stores the value in the variable(s) given in the second argument.

```
int scanf (format specifier, variable(s));
```

or

```
int scanf (format specifier, v1, v2, ....);
```

For instance, you can read a single character from the keyboard (same as `getchar()`) by using `scanf("%c", &c);` Here, `%c` is the conversion specifier. That is, it directs the conversion of the next input field. Normally, any conversion specifier starts with a `%` character followed by an appropriate character for that type – in this case it is a character. Table 1.3 shows the various conversion specifiers for `scanf()` function.

Table 1.3 Scanf() Conversion Specifiers

1	%c	Single character
2	%d or %i	Signed decimal integer
3	%e, %f, or %g	Floating point number
4	%o or %O	Octal integer
5	%x or %X	Hexadecimal integer
6	%s	Character string
7	%u	Unsigned character integer
8	%p	Pointer

The conversion characters `d`, `i`, `o`, `u` and `x` may be preceded by the character `l` (*ell*) to indicate that it is *long*.

`%ld` - *long int*eger (decimal)
`%lx` - *long* hexadecimal

Similarly, if `l` is used with `e`, `f` or `g`, then it indicates double.

`%lf` - double rather than *float*
`%le` - double rather than *float*
`%lg` - double rather than *float*

Few examples are shown below:

```
scanf("%d", &n); /* read an integer */
scanf("%f", &amount); /* reads a float */
scanf("%c", &c); /* reads a single character */
scanf("%ld", &long_int); /* reads a long integer */
scanf("%lf", &double_num); /* reads a double */
```

You can also read more than one variable using a single `scanf()` function. For example,

```
scanf("%d%d", &day, &year);
scanf("%d%f", &accno, &amount);
```

The Address Operator (&)

The ampersand(&) sign in the `scanf()` is called as an **address operator**. When `scanf()` function is encountered in the program, the control is passed to the `scanf()` routine, where it gets the characters from the keyboard and passes that to the calling program (i.e. `main()` function). For the moment, just remember that & must be used whenever you read a scalar variable from the keyboard or from a file.

Conversion modifier (*)

When * appears in the format specifier, it is called as a **suppression operator**. For example,

```
scanf("%*d%d", &value);
```

When two integers are entered, only the second value will be assigned to the variable. The advantage of * is mainly in processing the data files. For example, if the data file contains several fields like

```
<idno> <name> <total_salary> <deductions> <net_salary>
1700   DEEPAK 12726.00       1027.00       11699.00
```

The processing function may want to access the <idno> and <net_salary> fields and in that case the * operator may be used to skip the rest of the fields.

To Read specified number of digits

When you want to read only certain number of digits from the input device (irrespective of the number of digits entered by the user), then you can use the following syntax:

```
scanf("%d%2d", &n1, &n2);
```

When you enter 1234 1234, the variable n1 gets 1234 and the variable n2 gets 12. This is because of `"%2d"`.

The Output function - printf()

The output function `printf()` sends the values of variables given in the second argument to the console.

```
int printf(format specifier, v1, v2, ...);
```

Look at the below example,

```
printf("%d", n1); /* prints the value of n1 */
```

if the value of n1 is 21, then 21 will appear on the console. Below are some more examples:

```
printf("hello!"); /* prints Hello! */
printf("\n"); /* newline */
printf("The Answer is = %d\n", data);
printf("c = %c", c); /* prints value of c */
printf("%f", salary);
```

It is possible to send the *floating* point numbers to the screen in a highly formatted form.

```
%<flag> <width>.<precision><length>
```

Table 1.4

Sl. No.	Flag	Description
1	-	left justify the item
2	+	show the sign
3	space	show the sign
4	0	pad with leading zeros
5	#	0 for octal, 0x for hex
Sl. No.	**Width**	**Description**
1	integer	minimum field width
Sl. No.	**Width**	**Description**
1	integer	number of digits to display for an integer. number of digits after the decimal point.

Examples for the integer numbers and *floa*ting points are shown below:

Integer Numbers

```
                          Output
                column--> 1 2 3 4 5 7

1) printf("%d", 7206);    7 2 0 6
2) printf("%7d", 7206);        7 2 0 6
3) printf("%2d", 7206);   7 2 0 6
4) printf("%07d", 7206);  0 0 0 7 2 0 6
```

Float Numbers

```
                          Output
                column-->       1 2 3 4 5 6 7 8 9

1) printf("%f", 726.42);        7 2 6 . 4 2 0

2) printf("%5.1f", 726.42);    7 2 6 . 4
3) printf("%9.3f", 726.42);        7 2 6 . 4 2 0
4) printf("%6.2e", 726.42);    7 . 2 6 e + 0 2
```

Escape sequences

There is a special character, *backslash* (\) which causes the C compiler to interpret the character followed by \ in a different manner. For example, '\n' causes the cursor to

move to *newline*. Table 1.5 gives the escape sequences that could be used in `printf()` functions.

Table 1.5 Escape sequences

Sl. No.	Character	Description
1	\a	alert (bell)
2	\b	backspace
3	\f	formfeed
4	\n	newline
5	\r	carriage return
6	\t	tab
7	\v	vertical tab
8	\\	backslash
9	\?	question mark
10	\'	single quote
11	\"	double quote
12	\000	octal number
13	\xhh	hexadecimal number

See below for few examples

```
/* prints Hello and cursor comes to next line      */
printf("Hello\n");
/* the value of n1 and n2 with 8 blanks in between */
printf("%d\t%d\n", n1, n2);
/* the value of n1 and n2 are printed              */
printf("%d\n%d\n", n1, n2); /* two different lines */
```

Program 1.9
Fahrenheit to Celsius.

```
#include <stdio.h>
void main()
{
      float fahren, cel;
      printf("Enter the temperature in Fahrenheit: ");
      scanf("%f", &fahren); /* get the data */
      cel = (5.0 / 9.0) * (fahren - 32.0);
      printf("%5.1f\t%6.1f\n", fahren, cel);
}
```

Sample Run

```
Enter the temperature in Fahrenheit: 96.4
35.8
```

1.6 Operators and Expressions

1.6.1 Introduction

The real power of C language is with its large set of **operators**. C obtains its fastness in program execution through the use of sophisticated operators. There are unary, binary and ternary operators in C.

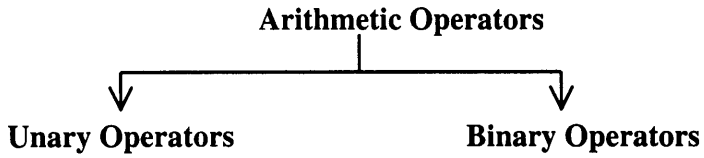
The concept of operator is same as what is used in mathematics. For example, + is an arithmetic operator which is used to add two given operands. Therefore, an operator is one which operates (performs a particular function – addition or subtraction, etc.) on operands. A set of operands and operators may be called as an expression. Expression such as $a + b * c$ have been in use for centuries.

There are various types of operators in C in which the major category include,

- arithmetic operators
- relational operators
- increment / decrement operators
- bit-wise operators
- assignment operators

1.6.2 Arithmetic Operators

You can perform arithmetic evaluation of expression with arithmetic operators.



Unary Operators

Unary Operators have only one operand. There are two unary operators + and -.

+	**positive sign**
-	**negative sign**

The unary operator – is to get the negative of the operand value and + does not do anything (i.e. you can write +5 or 5).

Example

-5, +4, +23.568, -67.9

Binary Operators

The binary operators have two operands appearing on either side of the operator. The following table gives you all the binary operators allowed in C language.

Sl. No.	Operator	Operation
1	+	Addition
2	-	Subtraction
3	*	Multiplication
4	/	Division
5	%	Modulus or remainder

The binary operators processes the operands given on either side (infix notation) and yields a single value. For example, 2 + 3 yields 5. The example Program 1.10 given below demonstrates the use of first four arithmetic operators.

Program 1.10
Arithmetic operators - demo

```
#include <stdio.h>
void main()
{
      int a = 5;
      int b = 2;
      float c = 7.6;
      float d = 2.7;
      int isum, idiff, imul, idiv;
      float fsum, fdiff, fmul, fdiv;
      isum = a + b;
      idiff = a - b;
      imul = a * b;
      idiv = a / b;
      fsum = c + d;
      fdiff = c - d;
      fmul = c * d;
      fdiv = c / d;
      printf("Integer arithmetic results: \n");
      printf("%d\t%d\t%d\t%d\n", isum, idiff,
                  imul, idiv);
      printf("Float arithmetic results: \n");
      printf("%f\t%f\t%f\t%f\n", fsum, fdiff,
                 fmul, fdiv);
}
```

Sample Run

```
Integer arithmetic results:
7       3       10      2
Float arithmetic results:
```