UNIT-2

Program development steps:

Program development contains following steps:

i). Creating the program
ii). Compiling the program
iii). Linking the program
iv). Executing the program

i) Creating the program:

In this step, the program can be written into a file through text editor. The file is saved on the disk with and extension. Corrections to the program at later stages are done to these editors. Once the program has written, it requires to be translated into machine language.

ii) Compiling the program:

This step is carried out by a program or compiler. Compiler translates the source code into object code.

Compilation of these cannot proceed successfully until and useless the source code is error free. Compiler generates messages if it encounters some errors in the source code. The error free source code translates in the object code and stored in a separate code with an extension .obj.

iii) Linking the program:

The linker links all the files and functions with the object code under execution.

Ex: printf-the linker links the user programs object with the object of the printf function. Now object code is ready for next phase.

iv) Execution the program:

The execution object code is loaded into the memory and the program execution begins. It encounters error to till the execution phase, Even though compilation phase is successful. These errors may be logical errors.

C language:

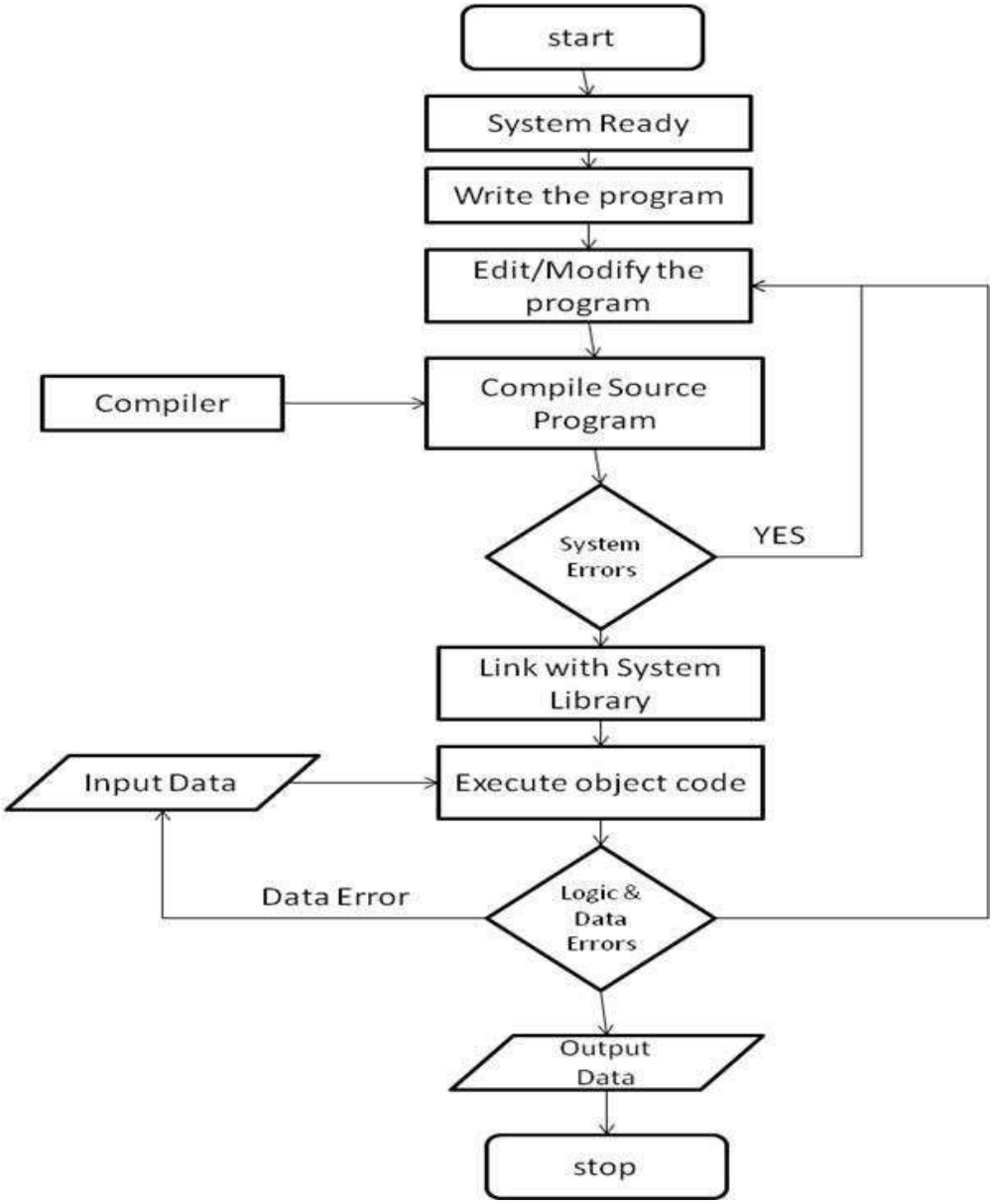
- C is case-sensitive language.
- C is a structured programming language.
- C is mother language.

Overview of "C":

- _C' seems a strange name for programming language. For this strange sounding language is one of the most popular words today.
- _C' was an off string of the —Basic Command Programming Language (BCPL)‖ called _B', developed in the 1960 at Cambridge University. _B' language was modified by Dennis ritchi and was implemented at bell laboratories in 1972. A new language was named _C'.
- Since it was developed along with be UNIX OS, it is strongly associated with UNIX. This OS, which was also developed at bell laboratory, was coded almost entirely in _C'.

Advantages of C languages:

- _C‘ is portable, means you say program write in one computer may run on another computer successfully.
- _C‘ is fast, means that the executable program obtained after compiling linking runs very fast.
- _C‘ is compact, means that the statements written in _C‘ language or generally sharp written on very powerful.
- _C‘ language is both simplicity of high level language, low level language, and middle level language.



Structure of C:

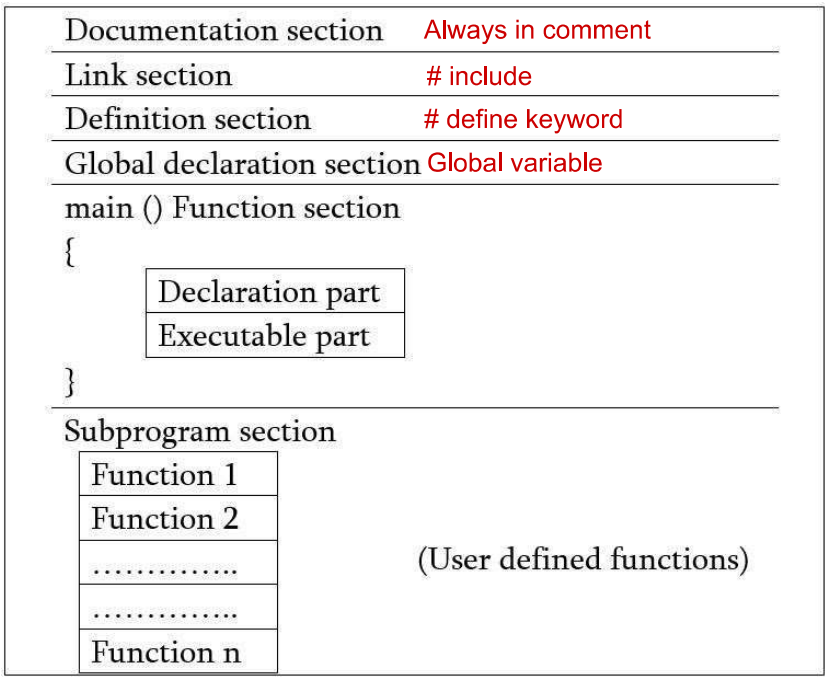


Fig: Structure of C

1. **Documentation section:** The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.
2. **Link section:** The link section provides instructions to the compiler to link functions from the system library such as using the *#include directive*.
3. **Definition section:** The definition section defines all symbolic constants such using the *#define directive*.
4. **Global declaration section:** There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the *user-defined functions*.
5. **main () function section:** Every C program must have one main function section. This section contains two parts; declaration part and executable part
    1. **Declaration part:** The declaration part declares all the *variables* used in the executable part.
    2. **Executable part:** There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The *program execution* begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.
6. **Subprogram section:** If the program is a *multi-function program* then the subprogram section contains all the *user-defined functions* that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order.

Character set:

Character set is a set of alphabets, letters and some special characters that are valid in C language.

Alphabets:

Uppercase: A B C ..... X Y Z

Lowercase: a b c ..... x y z

Digits:

0 1 2 3 4 5 6 7 8 9

Special Characters:

,	<	>	.	_
(	)	;	$	:
%	[	]	#	?
_	&	{	}	—
^	!	*	/	|
-	/	~	+	

White space Characters, blank space, new line, horizontal tab, carriage return and form feed

Keywords:

Keywords are predefined; reserved words used in programming that have special meaning. Keywords are part of the syntax and they cannot be used as an identifier. For example:

```
int money;
```

Here, **int** is a keyword that indicates **'money'** is a variable of type integer.

As C is a case sensitive language, all keywords must be written in lowercase. Here is a list of all keywords allowed in ANSI C.

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

Along with these keywords, C supports other numerous keywords depending upon the compiler. All these keywords, their syntax and application will be discussed in their respective topics

Identifiers:

Identifiers are the names you can give to entities such as variables, functions, structures etc. identifier names must be unique. They are created to give unique name to a C entity to identify it during the execution of a program. For example:

```
int money;
double balance;
```

Here, **money** and b**alance** are identifiers.

Also remember, identifier names must be different from keywords. You cannot use **int** as an identifier because **int** is a keyword.

Rules for writing an identifier:

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscore only. For ex., a variable named _NUMBER is not the same as the variable named _number‘ and neither of them is the same sa the variable named _Number‘. All three refer to different variables.
2. The underscore should **not** be used as the first character of a variable name because several compiler-defined identifiers in the standard C library have the underscore for beginning character, and it can conflict with system names.
3. A numeric digit should **not** be used as the first character of an identifier.
4. There is no rule on the length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different.
5. The identifier should not be a keyword of C.

The main() Function:

All C language programs must have a main() function. It‘s the core of every program. It‘s required. The main() function doesn‘t really have to do anything other than be present inside your C source code. Eventually, it contains instructions that tell the computer to carry out whatever task your program is designed to do. But it‘s not officially required to do anything. When the operating system runs a program in C, it passes control of the computer over to that program. This is like the captain of a huge ocean liner handing you the wheel. Aside from any fears that may induce, the key point is that the operating system needs to know where inside your program the control needs to be passed. In the case of a C language program, it‘s the main() function that the operating system is looking for.

At a minimum, the main() function looks like this:

```
main ( )
{
}
```

Like all C language functions, first comes the function‘s name, main, then comes a set of parentheses, and finally comes a set of braces, also called curly braces.

If your C program contains only this line of code, you can run it. It won‘t do anything, but that‘s perfect because the program doesn‘t tell the computer to do anything. Even so, the operating system found the main() function and was able to pass control to that function which did nothing but immediately return control right back to the operating system. It‘s a perfect, flawless program.

The set of parentheses after a C language function name is used to contain any arguments for the function — stuff for the function to digest. For example, in thesqrt() function, the parentheses hug a value; the function then discovers the square root of that value.

The main ( ) function uses its parentheses to contain any information typed after the program name at the command prompt. This is useful for more advanced programming. Beginning programmers should keep in mind what those parentheses are there for, but you should first build up your understanding of C before you dive into that quagmire.

The braces are used for organization. They contain programming instructions that belong to the function. Those programming instructions are how the function carries out its task or does its thing.

By not specifying any contents, as was done for the main() function earlier, you have created what the C Lords call a dummy function — which is kind of appropriate, given that you‘re reading this at Dummies.com.

The printf ( ) Function:

It is a library function. It is provided by the compiler, ready for use. In addition to its variable, a function, including main ( ), may optionally have arguments those are listed in the parenthesis following the function name.

In the C Programming Language, the **printf function** writes a formatted string to the *stdout* stream.

Syntax:

The syntax for the printf function in the C Language is:

printf(—Control_String‖ ,

list_of_variables); **Ex:-** printf(—Hello‖);

When the printf ( ) function is executable its built-in instructions process this argument. The result is that the string is display on the output device, usually assumed as a display screen terminal.

The output is

Hello

- C uses a semicolon as a statement terminator; the semicolon is required as a signal to the compiler to indicate that a statement is complete.
- All program instructions, which are also called statements, have to be written in lowercase characters.
- The following statement implies the preprocessor directive:
  #include<stdio.h>
- The #include directive includes the contents of a file during compilation. In this case the file stdio.h is added in the source program before the actual compilation begins.

Escape sequence:

The \n (pronounced backslash n) in the string argument of the function printf().

printf(―Welcome\nComputer‖);

is an example of escape sequence.

It is used print the new line character. If the program is executed, the \n does not appear in the output. Each \n in the string argument of a prinf() causes the cursor to be placed at the beginning of the next line of output.

Sno	Code	meaning
1	\a	Ring terminal bell (a is for alert)
2	\?	Question mark
3	\b	Backspace
4	\r	Carriage return
5	\f	Form feed
6	\t	Horizontal tab
7	\v	Vertical tab
8	\o	ASCII null character
9	\\	Back slash
10	\‖	Double quote
11	\‘	Single quote
12	\n	New line
13	\o	Octal constant
14	\x	Hexadecimal constant

Format Modifiers:

Describes the output as well as provides a placeholder to insert the formatted string. Here are a few examples:

Format	Explanation	Example
%c	Display Character	m
%s	Display group of characters	hai
%d (%i)	Display an integer	10
%ld	Display long decimal	51200
%o	Display Octal value	12
%u	Display unsigned integer	15
%x	Display hexa value	b
%X	Display HEXA value	B
%f	Displays a floating-point number in fixed decimal format	10.500000
%.1f	Displays a floating-point number with 1 digit after the decimal	10.5

Format	Explanation	Example
%e	Display a floating-point number in exponential (scientific notation)	1.050000e+01
%g	Display a floating-point number in either fixed decimal or exponential format depending on the size of the number (will not display trailing zeros)	10.5

COMMENT:

A "comment" is a sequence of characters beginning with a forward slash/asterisk combination (/*) that is treated as a single white-space character by the compiler and is otherwise ignored. A comment can include any combination of characters from the represent able character set, including newline characters, but excluding the "end comment" delimiter (*/). Comments can occupy more than one line but cannot be nested.

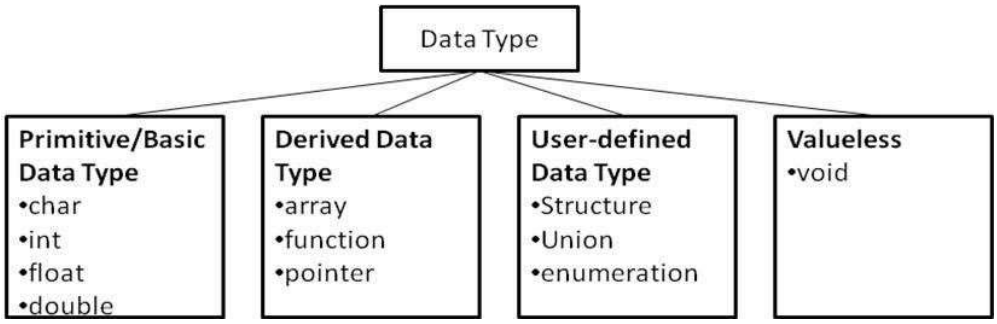
Comments can appear anywhere a white-space character is allowed. Since the compiler treats a comment as a single white-space character, you cannot include comments within tokens. The compiler ignores the characters in the comment.

Use comments to document your code. This example is a comment accepted by the compiler:

```
/* Comments can contain keywords such as
   for and while without generating errors. */
```

Data types of C:

Data types in c 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.



a) Primitive/Basic Data Type:

Integer data types:

C offers 3 different individual data types they are int, short int, long int. the difference between these 3 individual types are bytes required and range of values.

TYPE	SIZE (bits)	RANGE
short int	16	-2^{15} to $2^{15}-1$
int	16	-2^{15} to $2^{15}-1$
long int	32	-2^{31} to $2^{31}-1$
unsigned short int	16	0 to $2^{16}-1$
unsigned int	16	0 to $2^{16}-1$
unsigned long int	32	0 to $2^{32}-1$

2 bytes

Float data types:

Like integer, float are divided into 3 different individual data types they are float, double, long double. The differences between these 3 individual types are bytes required and range of values.

TYPE	SIZE (bits)	RANGE
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932 to 3.4E+4932

Character data types:

Keyword **char** is used for declaring character type variables.

For example: char a = ‗m‘;

TYPE	SIZE (bits)	RANGE
char	8	-128 to 127
unsigned char	8	0 to 255

b) User Defined Data Type:

The user defined data types are two types. They are

1. Type definition
2. Enumerated data type

1. Type definition:

The users can define and identifies that represents an existing data types the users as type definition.

Ex: typedef int sno;

sno c1,c2;

2. Enumerated data type:

User defined can be used to declare variables that can have one of the values out of many enumerations constant. After that the user can declare variables to be of this new type.

```
#include<stdio.h>
enum week{ sunday, monday, tuesday, wednesday, thursday,
friday, saturday}; main ( ){
  enum week today;
  today=wednesday;
  printf("%d day",today);
}
```

OPERATORS:

An operator is a symbol which represents a particular operation that can be performed on data.

The data itself (which can be either a variable or constant) is called operand. Expressions made by combining operators and operands.

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Unary operators
5. Bitwise operators
6. Logical/Boolean operators
7. Conditional operators
8. Special operators

1. Arithmetic operations:

The arithmetic operators in _C' language are + (addition), - (subtraction), * (multiplication), / (division), % (modulus) these operators are called arithmetic/binary operators. Each operand can be int, float, char.

Ex: x+y, x-y, x*y, x/y, x%y.

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = 10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0

2. Assignment operators:

These are used to assign the result of the expression to a variable, assignment operator is _=' Syntax: V op= Exp

V- variable
op-arithmetic operator
Exp-Expression

+= add assignment
-= minus assignment
*= multiply assignment
/= divide assignment
%= Modulus assignment

3. Relational operators:

The relational operators are used to test or compare the values between two operands. The relational or equality operators produce an integer value to express the condition of the comparison. If the condition is false then the integer result is _0‘. Otherwise, the integer result is nonzero.

<	less than	= =	equal to
< =	less than or equal to	! =	not equal to
>	greater than	=	assignment
> =	greater than or equal to		

4. Unary operators:

The unary _++‘ and _- -_operators increment or decrement the value in a variable by 1. There are _pre‘ and _post‘ variants for both operators that do slightly different things as explained below.

Var ++ increment _post‘ variant
++Var increment _pre‘ variant
Var - - decrement _post‘ variant
- -Var decrement _pre‘ variant

i) x=a++; → x=a; a=a+1;
ii) x=a- -; → x=a; a=a-1;
iii) x=++a; → a=a+1; x=a;
iv) x=- -a; → a=a-1; x=a;

5. Bitwise operators:

A smallest element in the memory on which they are able to operator is a bit, C suppose several bitwise operators.

& Bitwise AND
| Bitwise OR
>> Bitwise Right Shift
<<Bitwise Left Shift
^ Bitwise Exclusive OR
~ Bitwise Negation

Ex: x=7, y=8
z=x & y; =0
Ex: x=7, y=8
z=x | y;=15
Ex: x=7, y=8
z=x ^ y; =15
Ex: x=7
z=x >> 3
Ex: x=7
z=x << 2
Ex: x=1
z= ~ x;

6. Logical/Boolean operators:

These operators are used to compare two or more operands. The logical operator is also called unary operators.

! Logical NOT

&& Logical AND

|| Logical OR

Example	Result
! (5= =5)	0
(5<6) && (6<5)	0
(5<6) || (6<5)	1

7. Conditional operators:

The conditional operator **?** and **:** are sometimes called ternary operator since it operands on three operands, and it is consider has IF-THEN-ELSE in C statement.

```
#include<stdio.h>
main ( )
{
        int a=7,b=8;
        printf ( (a<b) ? "a is big" :  "b is big"       );
}
```

8. Special operators:

Some commonly used special operators are comma operator (**,**), sizeof operator, address operator (**&**) and value of address operator (*****).

i) Comma operator:

The operator permits two different expressions, appears in a situation where only one expression put ordinary be use. The expressions are separated by comma operator. Ex: int a, b;

int c = (a=10, b=20, a+b);

ii) Sizeof operator:

The size of operator returns the number of bytes occupied in memory by operand may be the variable or constant or datatype.

Ex: sizeof (int);

iii) Address operator:

The address of the operator (**&**) returns the address of the variable.

Ex: printf (— Address is %d ‖ , &a);

iv) Value of address operator:

The value at address operator (*) return the value stored at particular address.

Ex: x = * n;

Ex:

```
#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 the 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
```

Variable:

A variable is an identifier for a memory location in which data can be stored and subsequently recalled. All the variables have two important attributes.

A data type that is established when the variable is defined, e.g., integer, real, character. Once defined, the type of a C variable cannot be changed.

A value can be changed by assigning a new value to the variable. The kind of values a variable can assume depends on its type. For e.g., an integer variable can only take integer values, e.g., 2, 100, -12, -14, 0, 4.

Declaration of variables:

Any programming language any variable used in the program must be declared before it is used. This declaration tells the compiler what the variable name and what type of data it is.

In C language a declaration of variable should be done in the declaration part of the program. A type declaration statement is usually written at the beginning of the program.

Syntax: <data_type> <var1>,<var2>,<var3>,……,<var n>;

Ex: int i, count;
float price, area;
char c;

Assigning values:

Values can be assigned to variables by using the assignment operator. An assignment statement employees that the value of the variable on the left of the equal to the value of quantity on the right.

Syntax: <data_type> variable_name = constant;

Ex: int i = 20;

Declaration of statements:

In computer programming, a **declaration** specifies properties of an identifier: it declares what a word (identifier) *means*. Declarations are most commonly used for functions, variables, constants and classes, but can also be used for other entities such as enumerations and type definitions.

Here are some examples of declarations that are not definitions, in C:

```
extern char example1;
extern int example2;
void example3(void);
```

Here are some examples of declarations that are definitions, again in C:

```
char example1; /* Outside of a function definition it will be initialized to zero. */
int example2 = 5;
void example3(void) { /* definition between braces */ }
```

Expression:

An expression is a combination of variables constants and operators written according to the syntax of C language. In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Some examples of C expressions are shown in the table given below.

Algebraic Expression	C Expression
a x b – c	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
(ab / c)	a * b / c
$3x^2 + 2x + 1$	3*x*x+2*x+1
(x / y) + c	x / y + c

Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form

Variable = expression;

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before evaluation is attempted.

Example of evaluation statements are

x = a * b – c
y = b / c * a
z = a – b / c + d;

The following program illustrates the effect of presence of parenthesis in expressions.

Ex:

```
main ()
{
        float a, b, c x, y, z;
        a = 9;
        b = 12;
        c = 3;
        x = a – b / 3 + c * 2 – 1;
        y = a – b / (3 + c) * (2 – 1);
        z = a – ( b / (3 + c) * 2) – 1;
        printf (—x = %fn‖,x);
        printf (—y = %fn‖,y);
        printf (—z = %fn‖,z);
}
```

Output:

```
x = 10.00
y = 7.00
z = 4.00
```

Operator Precedence:

This page lists C operators in order of precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity
( ) [ ] . -> ++ --	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of *type*) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right
Note 1: Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are evaluated from inner to outer.		

Note 2:

Postfix increment/decrement have high precedence, but the actual increment or decrement of the operand is delayed (to be accomplished sometime before the statement completes execution). So in the statement **y = x * z++;** the current value of **z** is used to evaluate the expression (*i.e.,* **z++** evaluates to **z**) and **z** only incremented after all else is done.

Type Conversions:

There are two kinds of type conversion we need to talk about: automatic or **implicit** type conversion and **explicit** type conversion.

1. Implicit Type Conversion

The operators we have looked at can deal with different types. For example we can apply the addition operator + to an int as well as a double. It is important to understand how operators deal with different types that appear in the same expression. There are rules in C that govern how operators convert different types, to evaluate the results of expressions.

For example, when a floating-point number is assigned to an integer value in C, the decimal portion of the number gets truncated. On the other hand, when an integer value is assigned to a floating-point variable, the decimal is assumed as .0.

This sort of implicit or automatic conversion can produce nasty bugs that are difficult to find, especially for example when performing multiplication or division using mixed types, e.g. integer and floating-point values. Here is some example code illustrating some of these effects:

```
#include <stdio.h>
int main() {
    int a = 2;
    double b = 3.5;
    double c = a * b;
    double d = a / b;
    int e = a * b;
    int f = a / b;
    printf("a=%d, b=%.3f, c=%.3f, d=%.3f, e=%d, f=%d\n",a, b, c, d, e, f);
    return 0;
}
```

Output:

a=2, b=3.500, c=7.000, d=0.571, e=7, f=0

2. Explicit Type Conversion

Type Casting

There is a mechanism in C to perform **type casting** that is to force an expression to be converted to a particular type of our choosing. We surround the desired type in brackets and place that just before the expression to be coerced. Look at the following example code:

```
#include <stdio.h>
#include <stdio.h>
int main() {
    int a = 2;
    int b = 3;
    printf("a / b = %.3f\n", a/b);
    printf("a / b = %.3f\n", (double) a/b);
    return 0;
}
```

Mathematical library functions:

The <math.h> header file contains the mathematical operations like trigonometric and other mathematic formats.

Function	Return type	Description
acos(d)	double	Return the $\cos^{-1}$ of d
asin(d)	double	Return the $\sin^{-1}$ of d
atan(d)	double	Return the $\tan^{-1}$ of d
atan2(d1,d2)	double	Return the $\tan^{-1}$ of d1/d2
ceil(d)	double	Return a value rounded upto the next higher integer
cos(d)	double	Return the cos of d
cosh(d)	double	Return the hyperbolic cos of d
exp(d)	double	Return e of the power of d
fabs(d)	double	Return absolute value of d
floor(d)	double	Return a value rounded downto the next lower integer
fmod(d1,d2)	double	Return the remainder of d1/d2 (with same sign as d1)
labs(l)	long int	Return the absolute value of l
log(d)	double	Return natural logarithm of d
log10(d)	double	Return natural logarithm (base 10) of d
pow(d1,d2)	double	Return d1 raised to the d2 power ($d1^{d2}$)
sin(d)	double	Return the sin of d
sinh(d)	double	Return the hyperbolic sin of d
sqrt(d)	double	Return square root of d
tan(d)	double	Return the tan of d
tanh(d)	double	Return the hyperbolic tan of d

Basic Screen and keyboard I/O in C:

C provides several functions that give different levels of input and output capability. These functions are, in most cases, implemented as routines that call lower-level input/output functions.

The input and output functions in C are built around the concept of a set standard data streams being connected from each executing data streams or files are opened by the operating system and are available to every C and assembler program to use without having to open or close the files. These standard files or streams are called

stdin: Connected to the keyboard
stdout: Connected to the screen
stderr: Connected to the screen

The following two DataStream‘s are also available on MSDOS-based computers, but not an UNIX or other multi-user-based operating systems.

stdaux: Connected to the first serial communication
stdprn: Connected to the first parallel printer port

i) Non –formatted Input and Output
ii) Formatted Input and Output

i) Non-formatted Input and Output:

Non-formatted input and output can be carried out by standard input-output library functions in C. these can handle one character at a time.

For the input functions, it reads the character. For the output functions, it prints the single character on the screen.

a) Single character input:

The getchar( ) input function reads an unsigned char from the input stream stdin. To read a single character from the keyboard, the general form of the statement used to call the getchar() function is given as follows:

char_variable_name = getchar( );

where char_variable_name is the name of a variable of type char. The getchar() input function receives the character data entered, through the keyboard, and places it in the memory location allotted to the variable char_variable_name.

Ex:

```
#include<stdio.h>
main()
{
        char ch;
        int a,b,c;
        printf(―Would you like perform addition or subtraction\?\n‖);
        printf(―Type A for addition and S for subtraction ‖);
        ch = getchar( );
```

```
        if(ch==‘A‘ || ch==‘a‘)
        {
                printf(—\nEnter a and b values: ‖);
                scanf(—%d %d‖, &a, &b);
                c = a+b;
                printf(—\n Addition is %d‖,c);
        }
        else if(ch==‘A‘ || ch==‘a‘)
        {
                printf(—\nEnter a and b values: ‖);
                scanf(—%d %d‖, &a, &b);
                c = a-b;
                printf(—\n Subtraction is %d‖,c);
        }
        else
        {
                Printf(—\nInvalid letter‖);
        }
}
```

Output1:

Would you like perform addition or subtraction? Type A for addition and S for subtraction A Enter a and b values: 5 6 Addition is 11

Output2:

Would you like perform addition or subtraction? Type A for addition and S for subtraction S Enter a and b values: 8 3 Subtraction is 5

b) Single character output:

The putchar( ) function is identical in description to the getchar( ) function except the following difference. putchar( ) writes a character to the stdout data stream.

putchar(char_variable_name);

Ex:

```
char ch;
Ch=getchar();
putchar(ch);
```

ii) Formatted Input and Output functions:

When input and output is required in a specified format the standard library functions scanf( ) and printf( ) are used. The scanf( ) function allows the user to input data in a specified format. It can accept data in a specified format. It can accept data of different data type. The printf( ) function allows the user to output data of different data types on the console in a specified format.

a) Formatted input:

C using scanf function for formatted input. We shall explore all of the options that are available for reading the formatted data with scanf function.

The general format is:

scnaf(—control_string‖, var1,var2,var3,…..,varn);

The control string specifies the field format in which the data is to be entered and the variables var1, var2, var3,…..,varn specify the address of locations where the data is Stored. Control string and variables separated by commas.

Ex:

```
#include<stdio.h>
main()
{
        Int a, b, c;
        printf(—\nEnter a and b values: ‖);
        scanf(—%d %d‖, &a, &b);
        c = a+b;
        printf(—\n Addition is %d‖,c);
}
```

b) Formatted output:

The printf( ) function do differe from the sort of functions that are created by the programmer as they can take a variable number of parameters.

The general format is:

printf(—control_string‖,variable1,variable2,…..,variable);

Ex:

- printf(—welcome‖);

  The above statement displays **welcome** only.

  printf(—sum is %d‖, s);

  The above format specifier %d means convert the next value to a signed decimal integer, and hence will print **sum =** and then the value passed by the variable named **s** as a decimal integer.