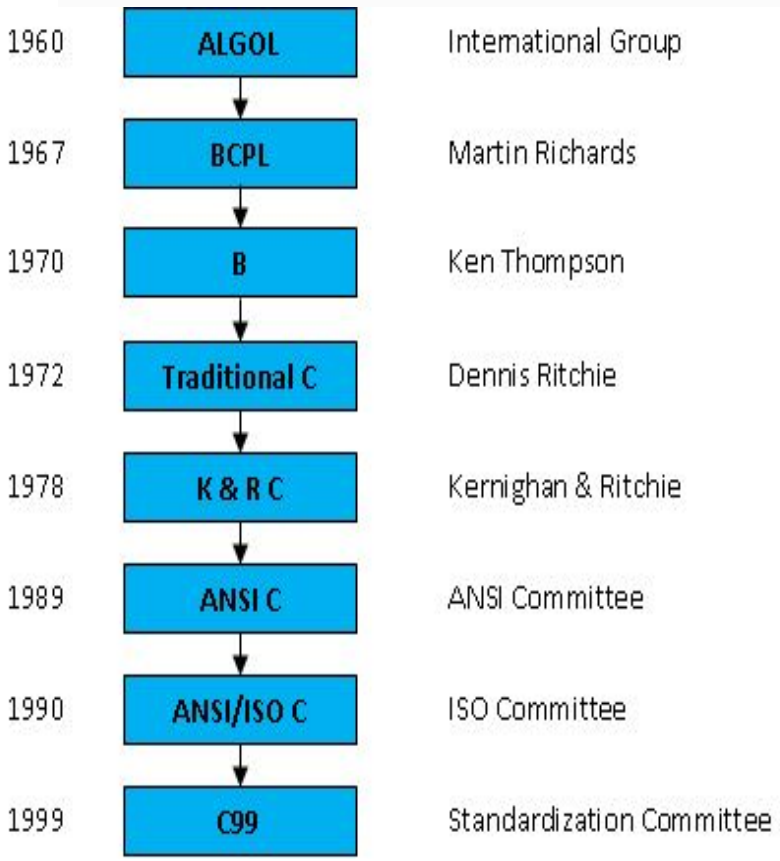
Chapter 2: Fundamentals of C

- C is a high level language.
- It also has the capability of low level programming. So, sometimes also referred as middle level Programming Language.
- Best Programming Language for learning procedural programming approach.
- It was developed at the Bell Laboratory, USA (now AT & T), in 1972.
- It is the outcome of Dennis Ritchie and Brian Kernighan.

History of C

- C is derived from two early programming languages such as BCPL (Basic Combined Programming Language) and **B** language developed by Ken Thompson.
- In 1972 Dennis Ritchie developed a new version of **B** and named it as **C**.
- In 1978, Dennis Ritchie and Brian Kernighan jointly published a detailed description of the C language document. It is known as **K & R C.**
- Some of Shortcomings of K & R C implementation are overcome by ANSI (American National Standard Institute) standards.



Characteristics of C

- C is a free-form language.
- C is a general purpose & structured programming language, which helps in development of system software.
- It has rich set of operators.
- It provides compact representation for expression.
- It allows manipulation of internal processor registers.
- Portability: any C program can be run in different machine with no modification.
- Rich set of data types and less number of reserved words.
- Pointer arithmetic and manipulation.
- Ability to extend itself by adding fuctions to its library

Program to print a simple message

```
#include<stdio.h>
void main()
{
        printf("hello world.");
}
```

Output

hello world.

Structure of a C program

```
preprocessor statement
global declaration;
main( )
{
local declaration;
executable statements;
}
user defined function
```

Preprocessor statements

- Begin with # symbol, also called the preprocessor directives.
- Direct the C preprocessor to include header files and also symbolic constants into C program. Few Statements :
- # include < stdio.h > : for standard input/output functions.
- # include "Test.h" : for file inclusion of the header files.
- # define NULL 0 : for defining symbolic constant, NULL=0

Local declarations::

- Variable Declaration: All Variables, array, function used in C program are declared and may be initialized with their basic data types.

Global declarations::

- Variable or function whose existence is known in main( ) function and other user defined function, are called the global variable.
- Its declaration is known as global declarations.
- Execution of C program starts with main( ).

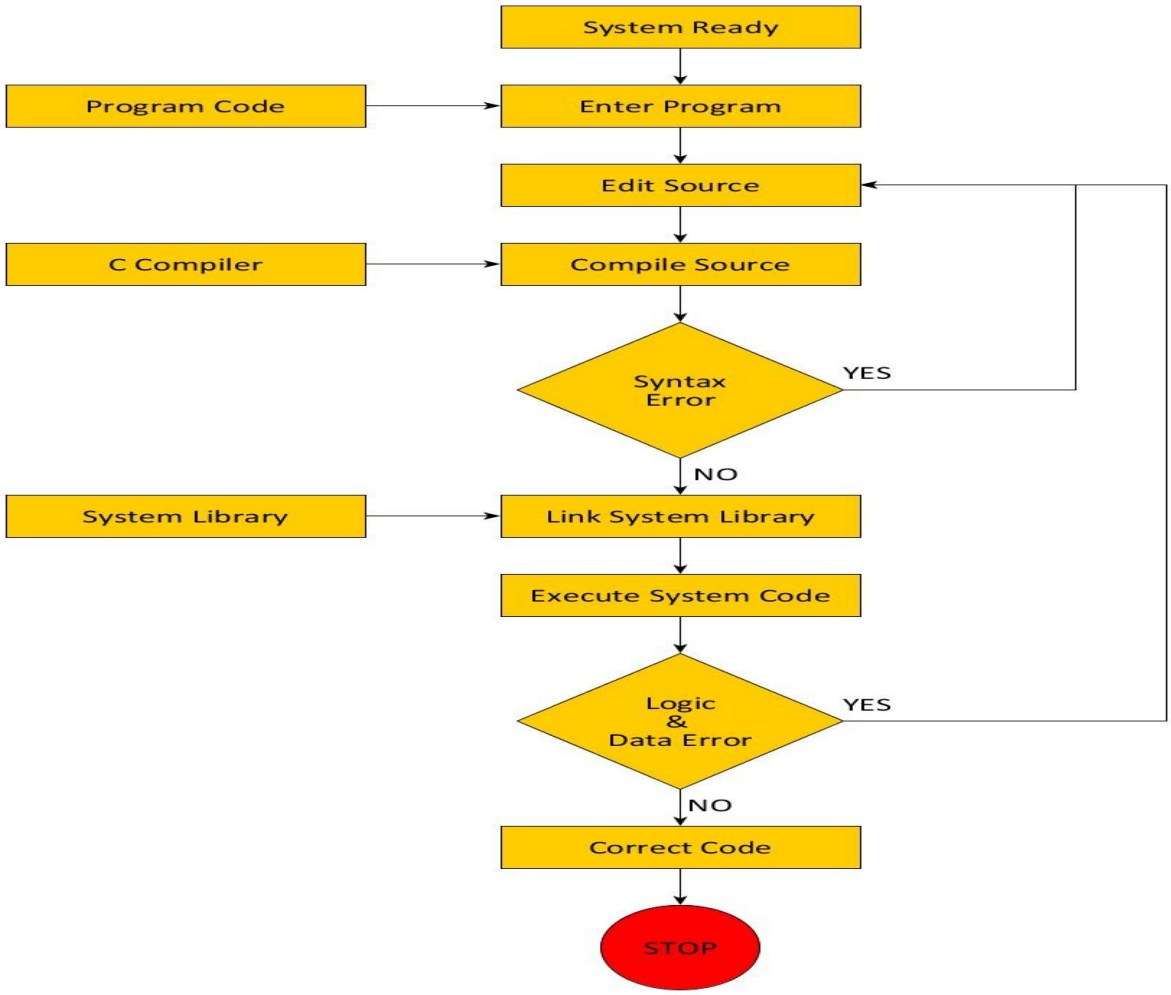
- Statements: the instruction to the computer to perform some specific task. They may be:

1. Arithmetic Statements.
2. Input/output Statements.
3. Control Statements.
4. Other Statements.

- Comments: explanatory notes on some instruction, which

are not executed and enclosed with in /* & */

Execution of C Program



Compiling and executing a C program

- Compiling a C program means translating it into object code. Compiler does that for C programs.
- Various tools are available which gives the facility of writing, editing, debugging and executing C language program.
- This type of environment is known as Integrated Development Environment (IDE).
- Examples : Turbo C, Borland C/C++, ANSI C and many more.

- There are basically five steps in the successful execution of a program:

1. Creating a program file (Source File).
2. Saving the program (with .c as extension).
3. Compilation.
4. Linking system library function.
5. Running (executing) the program.

Programming style

- Programming style is a set of rules or guidelines used when writing the source code for a computer program.
- It is often claimed that following a particular programming style will help programmers to read and understand source code conforming to the style, and help to avoid introducing errors.
- A classic work on the subject was The Elements of Programming Style, written in the 1970s, and illustrated with examples from the Fortran and PL/I languages prevalent at the time.
- The programming style used in a particular program may be derived from the coding conventions of a company or other computing organization, as well as the preferences of the author of the code.
- Programming styles are often designed for a specific programming language (or language family): style considered good in C source code may not be appropriate for BASIC source code, and so on. However, some rules are commonly applied to many languages.

Chapter 3: Data types in C

C character set :

Whenever one write any C program then it consists of different statements. Each C Program is set of statements and each statement is set of different c programming leximes. In C Programming each and every character is considered as single lexim. i.e [Basic Lexical Element]

Character Set Consists Of –

Types	Character Set
Lowercase Letters	a-z
Uppercase Letters	A to Z
Digits	0-9
Special Characters	!@#$%^&*
White Spaces	Tab Or New line Or Space

Valid C Characters : Special Characters are listed below –

Symbol	Meaning

~	Tilde
!	Exclamation mark
#	Number sign
$	Dollar sign
%	Percent sign
^	Caret
&	Ampersand
*	Asterisk
)	Right parenthesis
(	Left parenthesis
—	Underscore
+	Plus sign
|	Vertical bar
\	Backslash
`	Apostrophe
–	Minus sign
=	Equal to sign
{	Left brace
}	Right brace
[	Left bracket
]	Right bracket
:	Colon
”	Quotation mark

;	Semicolon
<	Opening angle bracket
>	Closing angle bracket
?	Question mark
,	Comma

Trigraph characters:

- Some of the characters like {}, [], \, |, ~ and ^ are missing in the above keyboard. Hence practically it may not be possible to write a C program using this keyboard.
- To solve this problem C suggested to use combination of 3 characters to produce a single character called trigraph character.
- A trigraph is a sequence of three characters, the first two of which are question marks
- C supports the following 9 trigraph characters.

Trigraph sequence	Equal character
??=	#
??(	[
??)	]
??/	\
??<	{
??>	}
??!	|
??’	^
??-	~

C Tokens Chart:

- In C Programming punctuation,individual words,characters etc are called tokens.
- Tokens are basic building blocks of C Programming

Token Example :

No	Token Type	Example 1	Example 2
1	Keyword	do	while
2	Constants	number	sum
3	Identifier	-76	89
4	String	“HTF”	“PRIT”
5	Special Symbol	*	@
6	Operators	++	/

Basic Building Blocks and Definition :

Token	Meaning
Keyword	A variable is a meaningful name of data storage location in computer memory. When using a variable you refer to memory address of computer
Constant	Constants are expressions with a fixed value
Identifier	The term identifier is usually used for variable names
String	Sequence of characters
Special Symbol	Symbols other than the Alphabets and Digits and white-spaces
Operators	A symbol that represent a specific mathematical or non mathematical action

Keywords in C Programming Language :

1. Keywords are those words whose meaning is already defined by Compiler
2. Cannot be used as Variable Name
3. There are 32 Keywords in C
4. C Keywords are also called as Reserved words .

32 Keywords in C Programming Language

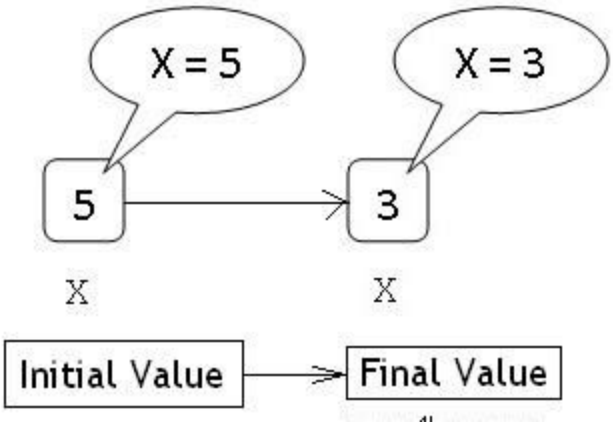
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Constant:

Constant in C means the content whose value does not change at the time of execution of a program.

Definition:

Constant means "**Whose value cannot be changed**"



Explanation :

- Initially 5 is Stored in memory location and name x is given to it
- After We are assigning the new value (3) to the same memory location

- This would Overwrite the earlier value 5 since memory location can hold only one value at a time
- The value of ‘3’,’5′ do not change ,so they are constants
- In Short the ‘Values of Constant Does not Change‘.

Different Types of C Constants :

Constant	**Type of Value Stored**
Integer Constant	Constant which stores integer value
Floating Constant	Constant which stores float value
Character Constant	Constant which stores character value
String Constant	Constant which stores string value

Declare Constant in C :

Declare constant of type integer then following two ways to declare it –

```
const int a = 1;
```

```
int const a = 1;
```

above declaration is bit confusing but no need to worry, We can start reading these variables from right to left. i.e

Declaration	Explanation
const int a = 1;	read as “a is an integer which is constant”

int const a = 1;	read as “a is a constant integer”

Real constants in c:

- A real constant must have at least one digit
- It must have a decimal point
- It could be either positive or negative
- If no sign precedes an integer constant, it is assumed to be positive.
- No commas or blanks are allowed within a real constant.

Single Character Constant :

1. Character Constant Can hold Single character at a time.
2. Contains Single Character Closed within a pair of Single Quote Marks
3. Single Character is smallest Character Data Type in C.
4. Integer Representation : Character Constant have Integer Value known as ‘ASCII’ value
5. It is Possible to Perform Arithmetic Operations on Character Constants

   Examples are:

1. ‘a’
2. ‘1’
3. ‘#’
4. ‘<‘
5. ‘X’

Way 1: Declaring Single Variable

```
char variable_name;
```

Way 2: Declaring Multiple Variables

```
char var1,var2,var3;
```

Way 3: Declaring & Initializing

```
char var1 = 'A',var2,var3;
```

Format Specifier for Character Variable :

- “%c” is used as format specifier for character inside C.
- However we can also use “%d” as format specifier because “Each Character have its equivalent intergervalue known as ASCII Value. “

Using Format Specifier to Print Character Variable :

Sample 1:

```
printf("%d",'a');   //Output : 97
```

Sample 2:

```
printf("%c",'97');  //Output : a
```

These two Represent the same Result

Example 1 : Creating Variable and Displaying Character Data

```
#include<stdio.h>
int main()
{
char cvar = 'A';
printf("Character is : %c",cvar);
return(0);
}
```

Output :

```
Character : A
```

String Constant in C Programming Language :

1. String is “Sequence of Characters“.
2. String Constant is written in Pair of Double Quotes.
3. String is declared as Array of Characters.
4. In C , String data type is not available.
5. Single Character String Does not have Equivalent Integer Value i.e ASCII Value

Different Constant Values Contained inside String :

String with Single Character

```
"a"
```

String With Multiple Characters

```
"Ali"
```

String With Digits

```
"123"
```

String With Blanks

```
"How are You"
```

Note :

```
1] 'A' is not Equal to "A"
2] 'A' ==> Requires 1 byte Memory
3] "A" ==> Requires 2 byte Memory
```

Summary :

Example	Meaning
“a”	String with Single Character
“Ali”	String With Multiple Characters
“123?	String With Digits
“How are You”	String With Blanks

Backslash character constants in c:

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.
- Given below is the list of special characters and their purpose.

Backslash_character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab

\”	Double quote
\’	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark
\N	Octal constant (N is an octal constant)
\XN	Hexadecimal constant (N – hex.dcml cnst)

Variables :

Variable in C Programming is also called as container to store the data. Variable name may have different data types to identify the type of value stored. Suppose we declare variable of type integer then it can store only integer values. Variable is considered as one of the building block of C Programming which is also called as identifier.

Rules for naming c variable:

1. Variable name must begin with letter or underscore.
2. Variables are case sensitive
3. They can be constructed with digits, letters.
4. No special symbols are allowed other than underscore.
5. sum, height, _value are some examples for variable name

Data Types

The C language is very rich in its data types. The variety of data types available allows the programmer to select an appropriate type according to the application or machine’s requirement. Data types are an extensive system used for declaring variables and functions of different types.

There are three classes of data types:

- Primary (fundamental) data types
- Derived Data types
- User Defined data types

Primary Data Types

Primary data types can be classified into basic and built-in types. These data types are the most basic building blocks of any programming language and numerous composite data types are constructed using them.

Some primary data types are

Integer Types:

This data type as the name suggests, is used to hold only integer values. Generally, an integer stores values in a range limited from -32768 to 32767. A signed integer uses one bit for sign and 15 bits for the magnitude of a number. The integer has three classes of storage both signed and unsigned. These are classified on the basis of range of values held by them:

- Short integer: -128 to 127
- Integer: -32768 to +32767
- Long Integer: -2,147,483,648 to 2,147,483,647

Syntax

```
short i=value;

int i=value;

long int i=value;
```

```
int i=3;

long int i =340000

short i=2;
```

Floating Point Types:

This data type stores real numbers in 32 bits, with 6 digits of precision. The keyword ‘float’ defines the floating-point data type. The double type is used when more accurate data is required. When more accurate data is required, the double type is used. Double type uses 64 bits giving a precision of 14 digits. Double type precision is more than the float type. Long double further extends the precision it uses by 80 bits.

Void Types:

This data type has no values and usually specifies function type. The void type function does not return any values. It can also play the role of generic type, which means it can represent any standard type.

Syntax:

```
public void display()
```

Character Types:

A single character in C is defined as character (char) type data. They are usually stored in 8 bits of storage and the character can be signed and unsigned. Range of characters can be from -128 to 127 signed and 0 to 255 under unsigned.

Syntax

```
char ch=’a’
```

User-Defined Data Types:

C supports the features “typedef” that allows users to define the identifier which would represent an existing data type. This defined data type can then be used to declare variables:

Syntax:

```
typedef int  numbers;



numbers  num1,num2;
```

In this example, num1 and num2 are declared as int variables. The main advantage of user defined data type is that it increases the program’s readability.

Another type is enumerated type. This is also a user defined data type

Syntax:

```
enum  identifier {value1,value2, value 3,...}
```

“Enum” is the keyword and “identifier” is the user defined data type that is used to declare the variables. It can have any value enclosed within the curly braces. For example:

```
enum day {January, February,March,April,..};



enum day month_st,month_end;
```

The compiler automatically assigns integer digits beginning from 0 to all the enumeration constants. For example, “January ” will have value 0 assigned, “February” value 2 assigned and so on. You can also explicitly assign the enumeration constants.

Derived Data Type:

These data types are formed by a combination of two or more primary data types. They have extended the scope of C language. The most common are pointers, arrays, union and structures.

DECLARING & INITIALIZING C VARIABLE:

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- Variable initialization means assigning a value to the variable.

Type	Syntax
Variable declaration	data_type variable_name; Example: int x, y, z; char flat, ch;
Variable initialization	data_type variable_name = value; Example: int x = 50, y = 30; char flag = ‘x’, ch=’l’;

Symbolic constant in C Language:

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 a C program consists of the following symbolic constant definitions.

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

#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.
The preprocessor statements begin with a #symbol, and are not end with semicolon. By convention, preprocessor constants are written in UPPERCASE.

Example: 1

```
#include<stdio.h>
#include<conio.h>
#define TRUE 1
#define PI 3.141593

void main()
  {
   float    a;
   float    b;
   float    c;
   float   d=PI;
   clrscr();
   if(TRUE)
      {
      a=100;
      b=a*10;
      c=b-a;
      }
   printf("\na=%f\nb=%f\nc=%f\nPI=%f",a,b,c,d);
```

```
 getch();
}
```

Declaring variable as constant

When the const qualifier is used, the declared variable must be initialized at declaration. It is then not allowed to be changed.

While the idea of a variable that never changes may not seem useful, there are good reasons to use const. For one thing, many compilers can perform some small optimizations on data when it knows that data will never change. For example, if you need the value of π in your calculations, you can declare a const variable of pi, so a program or another function written by someone else cannot change the value of pi.

Note that a Standard conforming compiler must issue a warning if an attempt is made to change a const variable - but after doing so the compiler is free to ignore the const qualifier.

Volatile

Volatile is a special type of modifier which informs the compiler that the value of the variable may be changed by external entities other than the program itself. This is necessary for certain programs compiled with optimizations – if a variable were not defined volatile then the compiler may assume that certain operations involving the variable are safe to optimize away when in fact they aren't. *volatile* is particularly relevant when working with embedded systems (where a program may not have complete control of a variable) and multi-threaded applications.

Overflow and underflow of data in 'c' language

Note:
The size of short is 2 bytes any where (maximum size of short is 2bytes)
The size of int is 2 bytes in some compilers and 4bytes in other compilers (minimum size of int is 2bytes)
The size of long double is 10 bytes in some compilers and 12bytes in other compilers

Various forms in declaring variables:

- signed short int x;
- short signed int x;
- short int x;
- signed int x;
- int x;
- short x;
- signed x;

- long signed int x;
- signed long int x;
- long int x;
- long x;

- unsigned short int x;
- short unsigned int x;
- unsigned int x;
- unsigned x;

Chapter 4: Operators and their Hierarchy

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression A + B * 5. where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.

- Operators in C programming
- Arithmetic Operators
- Increment and Decrement Operators

- Assignment Operators
- Relational Operators
- Logical Operators
- Conditional Operators
- 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)

- Example #1: Arithmetic Operators

```
// C Program to demonstrate the working of arithmetic operators

#include <stdio.h>
int main()
{
   int a = 9,b = 4, c;
   c = a+b;
   printf("a+b = %d \n",c);
   c = a-b;
   printf("a-b = %d \n",c);
   c = a*b;
   printf("a*b = %d \n",c);
   c=a/b;
   printf("a/b = %d \n",c);
   c=a%b;
```

```
    printf("Remainder when a divided by b = %d \n",c);
    return 0;
}
```

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
```

Remainder when a divided by b=1 computes the remainder. When a = 9 is divided by b = 4, the remainder is 1. The % operator can only be used with integers.

Suppose a = 5.0, b = 2.0, c = 5 and d = 2. Then in C programming,

```
a/b = 2.5  // Because both operands are floating-point variables
a/d = 2.5  // Because one operand is floating-point variable
c/b = 2.5  // Because one operand is floating-point variable
c/d = 2    // Because both operands are integers
```

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.

Example #2: Increment and Decrement Operators

```
// C Program to demonstrate the working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
```

```
    float c = 10.5, d = 100.5;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);
    return 0;
}
```

Output

```
++a = 11
--b = 99
++c = 11.500000
++d = 99.500000
```

Here, the operators ++ and -- are used as prefix. These two operators can also be used as postfix like a++ and a—

Suppose, a = 5 then,

```
++a;         //a becomes 6
a++;         //a becomes 7
--a;        //a becomes 6
a--;        //a becomes 5
```

Important difference should know when these two operators are used as prefix and postfix.

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

Example #3: Assignment Operators

```
// C Program to demonstrate the working of assignment
operators
#include <stdio.h>
int main()
{
   int a = 5, c;
   c = a;
   printf("c = %d \n", c);
   c += a; // c = c+a
   printf("c = %d \n", c);
   c -= a; // c = c-a
   printf("c = %d \n", c);
   c *= a; // c = c*a
   printf("c = %d \n", c);
   c /= a; // c = c/a
   printf("c = %d \n", c);
   c %= a; // c = c%a
   printf("c = %d \n", c);
   return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

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

Example #4: Relational Operators

```
// C Program to demonstrate the working of arithmetic
operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;
    printf("%d == %d = %d \n", a, b, a == b); // true
    printf("%d == %d = %d \n", a, c, a == c); // false
    printf("%d > %d = %d \n", a, b, a > b); //false
    printf("%d > %d = %d \n", a, c, a > c); //false
   printf("%d < %d = %d \n", a, b, a < b); //false
    printf("%d < %d = %d \n", a, c, a < c); //true
   printf("%d != %d = %d \n", a, b, a != b); //false
    printf("%d != %d = %d \n", a, c, a != c); //true
    printf("%d >= %d = %d \n", a, b, a >= b); //true
    printf("%d >= %d = %d \n", a, c, a >= c); //false
    printf("%d <= %d = %d \n", a, b, a <= b); //true
    printf("%d <= %d = %d \n", a, c, a <= c); //true
```

```
    return 0;
}
```

Output

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

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
&&	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.

Example #5: Logical Operators

```
// C Program to demonstrate the working of logical operators
```

```
#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10, result;
   result = (a = b) && (c > b);
   printf("(a = b) && (c > b) equals to %d \n", result);
   result = (a = b) && (c < b);
   printf("(a = b) && (c < b) equals to %d \n", result);

   result = (a = b) || (c < b);
   printf("(a = b) || (c < b) equals to %d \n", result);
   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) equals to %d \n", result);
   result = !(a != b);
   printf("!(a == b) equals to %d \n", result);
   result = !(a == b);
   printf("!(a == b) equals to %d \n", result);
   return 0;
}
```

Output

```
(a = b) && (c > b) equals to 1
(a = b) && (c < b) equals to 0
(a = b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

Explanation of logical operator program

(a = b) && (c > 5) evaluates to 1 because both operands (a = b) and (c > b) is 1 (true).

(a = b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).

(a = b) || (c < b) evaluates to 1 because (a = b) is 1 (true).

(a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).

!(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).

!(a == b) evaluates to 0 because (a == b) is 1 (true).
Hence, !(a == b)is 0 (false).

Bitwise Operators

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

The sizeof operator

The sizeof is an unary operator which returns the size of data (constant, variables, array, structure etc).

Example #6: sizeof Operator
```
#include <stdio.h>
int main()
```

```
{
   int a, e[10];
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));
   printf("Size of integer type array having 10 elements = %lu
bytes\n",       sizeof(e));
   return 0;
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
Size of integer type array having 10 elements = 40 bytes
```

C Ternary Operator (?:)

A conditional operator is a ternary operator, that is, it works on 3 operands.
Conditional Operator Syntax
conditionalExpression ? expression1 : expression2
The conditional operator works as follows:
The first expression conditionalExpression is evaluated at first. This expression evaluates to 1 if it's and evaluates to 0 if it's false.
If conditionalExpression is true, expression1 is evaluated.
If conditionalExpression is false, expression2 is evaluated.

Example #7: C conditional Operator

```
#include <stdio.h>
int main(){
   char February;
   int days;
   printf("If this year is leap year, enter 1. If not enter any
integer: ");
   scanf("%c",&February);
   // If test condition (February == 'l') is true, days equal to 29.
   // If test condition (February =='l') is false, days equal to 28.
   days = (February == '1') ? 29 : 28;
   printf("Number of days in February = %d",days);
   return 0;
}
```

Output

If this year is leap year, enter 1. If not enter any integer: 1

Number of days in February = 29

C Type Conversion – Implicit & Explicit Type Conversion in C

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:

Implicit Type Conversion

When the type conversion is performed automatically by the compiler without programmers 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.

The sequence of rules that are applied while evaluating expressions are given below:

All short and char are automatically converted to int, then,

If either of the operand is of type long double, then others will be converted to long double and result will be long double.

Else, if either of the operand is double, then others are converted to double.

Else, if either of the operand is float, then others are converted to float.

Else, if either of the operand is unsigned long int, then others will be converted to unsigned long int.

Else, if one of the operand is long int, and the other is unsigned int, then

if a long int can represent all values of an unsigned int, the unsigned int is converted to long int.

otherwise, both operands are converted to unsigned long int.

Else, if either operand is long int then other will be converted to long int.

Else, if either operand is unsigned int then others will be converted to unsigned int.

It should be noted that the final result of expression is converted to type of variable on left side of assignment operator before assigning value to it.

Also, conversion of float to int causes truncation of fractional part, conversion of double to float causes rounding of digits and the conversion of long int to int causes dropping of excess higher order bits.

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.

For example,

1 x=(int)a+b*d;

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:
All integer types to be converted to float.
All float types to be converted to double.
All character types to be converted to integer.

C Operator Precedence Table

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	right-to-left

%= &= ^= |= <<= > >=	Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	
,	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.
See postinc.cfor another example.

Chapter 5: Input/ Output functions

Introduction

C programming has several in-built library functions to perform input and output tasks.

Two commonly used functions for I/O (Input/Output) are printf() and scanf().

The scanf() function reads formatted input from standard input (keyboard) whereas the printf() function sends formatted output to the standard output (screen).

The printf() is a library function to send formatted output to the screen. The printf() function is declared in "stdio.h" header file.

stdio.h is a header file (standard input output header file) and #include is a preprocessor directive to paste the code from the header file when necessary. When the compiler encounters printf() function and doesn't find stdio.h header file, compiler shows error.

Reading character

Format string %c is used in case of character types.

When a character is entered in the above program, the character itself is not stored. Instead, a numeric value(ASCII value) is stored.

And when displayed that value using "%c" text format, the entered character is displayed.

Example #: C ASCII Code

```
#include <stdio.h>
int main()
{
```

```
    char chr;
    printf("Enter a character: ");
    scanf("%c",&chr);         // When %c text format is used,
character is displayed in case of character types
    printf("You entered %c.\n",chr);   // When %d text format is
used, integer is displayed in case of character types
    printf("ASCII value of %c is %d.", chr, chr);
    return 0;
}
```

Output
Enter a character: g
You entered g.
ASCII value of g is 103.
The ASCII value of character 'g' is 103. When, 'g' is entered, 103 is stored in variable var1 instead of g.

Display a character’s ASCII code of that character. This is shown by following example.
Example #6: C ASCII Code

```
#include <stdio.h>
int main()
{
    int chr = 69;
    printf("Character having ASCII value 69 is %c.",chr);
    return 0;
}
```

Output
Character having ASCII value 69 is E.

getchar() & putchar() functions

The getchar() function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one characters.
The putchar() function prints the character passed to it on the

screen and returns the same character. This function puts only single character at a time. In case you want to display more than one characters, use putchar() method in the loop.

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

When compiled the above code will ask to enter a value. Enter the value, it will display the value you have entered.

gets() & puts() functions

The gets() function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF (end of file). The puts() function writes the string s and a trailing newline to stdout.

```
#include<stdio.h>
void main()
{
 char str[100];
 printf("Enter a string");
 gets( str );
 puts( str );
 getch();
}
```

When you will compile the above code,it will ask you to enter a string. When you will enter the string, it will display the value you have entered.

Below are functions applied on characters using <ctype.h> library function.

Functions	Description
isalpha()	checks whether character is alphabetic
isdigit()	checks whether character is digit
isalnum()	Checks whether character is alphanumeric
isspace()	Checks whether character is space
islower()	Checks whether character is lower case
isupper()	Checks whether character is upper case
isdigit()	Checks whether character is hexadecimal
iscntrl()	Checks whether character is a control character
isprint()	Checks whether character is a printable character
ispunct()	Checks whether character is a punctuation
isgraph()	Checks whether character is a graphical character

tolower( )	Checks whether character is alphabetic & converts to lower case
toupper ()	Checks whether character is alphabetic & converts to upper case

```c
/* Fig 9.21: fig09_21.c */
/* Using a scan set */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   char z[ 9 ]; /* define array z */

   printf( "Enter string: " );
   scanf( "%[aeiou]", z ); /* search for set of characters */

   printf( "The input was \"%s\"\n", z );
   return 0; /* indicates successful termination */
} /* end main */
```

Output

Enter String: hahahaha

The input was: aaaa

%[aeiou] causes scanf to input only character listed in [].

Inputting/Outputting Integer Numbers:

Format : % w d

w-integer number, d-data type,

Example #: C Integer Output

```
#include <stdio.h>
int main()
{
   int testInteger = 5;
   printf("Number = %d", testInteger);
```

```
    return 0;
}
```
Output
Number = 5
Inside the quotation of printf() function, there is a format string "%d" (for integer). If the format string matches the argument (testInteger in this case), it is displayed on the screen.

Example #3: C Integer Input/Output

```
#include <stdio.h>

int main()

{

    int testInteger;

    printf("Enter an integer: ");

    scanf("%d",&testInteger);

    printf("Number = %d",testInteger);

    return 0;

}
```

Output

Enter an integer: 4

Number = 4

Inputting/outputting Real Numbers:

Format : % w sd

w-integer number, d-data type, s-points after decimal

Example #: C Floats Input/Output

```
#include <stdio.h>
int main()
{
```

```
    float f;
    printf("Enter a number: ");
// %f format string is used in case of floats
    scanf("%f",&f);
    printf("Value = %f", f);
    return 0;
}
```

Output

```
Enter a number: 23.45
Value = 23.450000
```

The format string "%f" is used to read and display formatted in case of floats.

More on Input/Output of floats and Integers

Integer and floats can be displayed in different formats in C programming.

Example #: I/O of Floats and Integers

```
#include <stdio.h>
int main()
{
    int integer = 9876;
    float decimal = 987.6543;    //  Prints the number right justified within 6 columns
    printf("4 digit integer right justified to 6 column: %6d\n", integer);      // Tries to print number right justified to 3 digits but the number is not right adjusted because there are only 4 numbers
    printf("4 digit integer right justified to 3 column: %3d\n", integer);       // Rounds to two digit places
    printf("Floating point number rounded to 2 digits: %.2f\n",decimal);     // Rounds to 0 digit places
    printf("Floating point number rounded to 0 digits: %.f\n",987.6543);    // Prints the number in exponential notation(scientific notation)
    printf("Floating point number in exponential form: %e\n",987.6543);
```

```
    return 0;
}
```

Output

```
4 digit integer right justified to 6 column:   9876
4 digit integer right justified to 3 column: 9876
Floating point number rounded to 2 digits: 987.65
Floating point number rounded to 0 digits: 988
Floating point number in exponential form: 9.876543e+02
```