Unit-2

C Language :

C is a programing language developed at AT&T's Bell laboratories of USA in 1972. It was designed & written by Dennis Ritchie.

C language is simple, more reliable, and easy to use.

In industry C language is still survived from last three decades.

Major part of popular ~~operation~~ operating system like windows, UNIX, Linux are still written in C.

The Character Set

Character denotes any alphabet, digit or special symbol used to represent information.

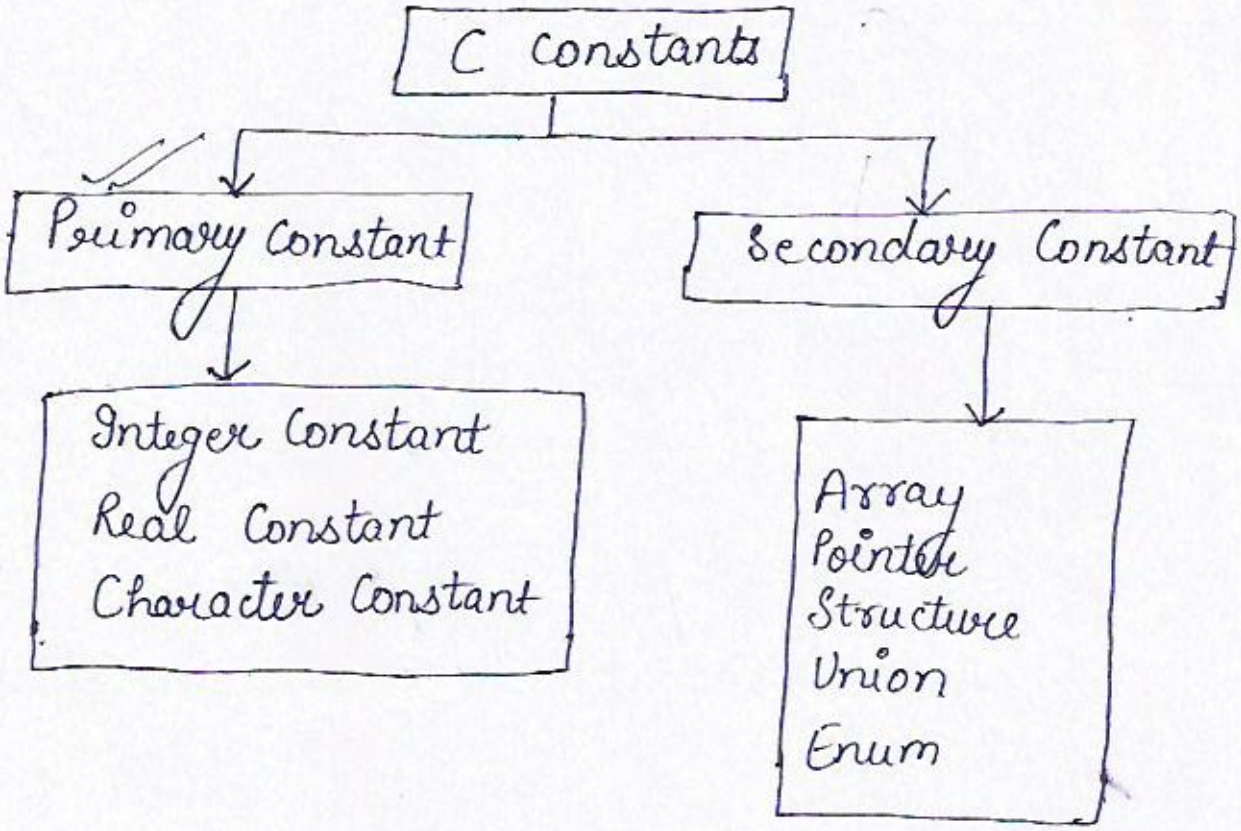
eg: Alphabets - A, B, ----- Y, Z
a, b, ----- y, z

Digits - 0, 1, 2, 3- - - - 9

Special Symbol - ! @ # % & ( ) - _ { }
[ ] : ; " ' ? $

Constants, Variable & Keywords

→ A constant is an entity that doesn't change during the entire source code.



Rules for Integer Constants

(a) An integer constant must have at least one digit.

(b) It must not have a decimal point

(c) It can be either positive or negative.

(d) If no sign precedes an integer constant, it is assumed to be positive

No commas or blanks are allowed within integer constant.

(f) The allowable range for integer constant is -2147483648 to +2147483647. (for Visual Studio & gcc compiler)

for Turbo C (-32768 to 32767).

eg 426, +782, -8023

Real Constants:

Real constants are often called as floating point constants. Real constant could be written in two forms – Fractional & Exponential.

Rules

(a) A real constant must have at least one digit
(b) It must have a decimal point.
(c) It could be either positive or negative.
(d) Default sign is positive
(e) No comma or blank are allowed within a real constant.

eg: 325.34
426.0
-32.79

In exponential form of representation, the real constant is represented in two parts. Part appearing before 'e' is called mantissa whereas the part following 'e' is called exponent

Thus 0.000342 can be represented as 3.4e-4.

Rules:

(a) The mantissa part & the exponential part should be seprated by letter e or E.

(b) The mantissa part may have positive or negative sign.

(c) Default sign of mantissa is positive

(d) The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.

(e) Range of Real constants expressed in exponential form is -3.4e38 to 3.4e38

eg +3.2e-5
4.1 e 8
-0.2E+3

Character Constants

* A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
  Both the inverted commas should point left.
* Maximum length of a character constant is 1
  eg 'A', '5' '='

Variables :

An entity that may vary during program execution is called variable.
Variable names are given to location of memory, These locations can contain integer, real or character constants.

In any language, the type of variables that it can support depend on the type of constant that it can handle.

For example an integer variable can hold only integer constant, Real variable can hold Real constant & so on.

Rules for Constructing Variables Name

(a) A variable name is any combination of 1 to 31 alphabets, digits or underscore.

(b) The first ~~alphabet~~ character in variable name is always a alphabet or underscore.

(c) No commas or blanks are allowed within variable name.

(d) No special symbol other than underscore (_) can be used in variable name.

eg: int a, si, m_1;
float rate;
char code

Keywords

→ Keywords are the words whose meaning has already been explained to the C compiler

→ These keywords cannot be used as variable name

→ Some time these keywords are also known as reserved keywords or "Reserved words"

There are 32 keywords reserved in C-language. ie

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

Data Type

Data Type indicate the type of data that a variable can hold. The data may be numeric, alphabatic, symbolic etc.

In C language the data type of any data is necessarily known in order to use them in program.

There are broadly two types of data types in C

→ Built in data types (Pre existing that are known to compiler)

→ User defined data types (These are defined by the

Built in Data Types

They are basic or primitive data type ~~of data that a variable can hold~~ and designate one single value. There are four basic fundamental built in data type in C

* Integer (int) size is 2 bytes
* Floating point (float) size is 4 bytes
* Double (double) size is 8 bytes
* Character (char) size is 1 byte.

Integer:

This is a keyword used to indicate an integer number which is sequence of digits without a decimal point.

The range of integer values lies between -32,768 to 32,767

eg -247, 14020, 27246, +1966

Invalid

3,333 (comma not allowed)
-34.0 (decimal point)
+3,486.30 (comma & decimal)
999999 (out of range)

Float:

This is a keyword used to indicate the floating point number. Floating point number are same as decimal number but they include fractional part as well.

eg -263.238
2.63238E+02 {2.63238×10^{2}}
2.63E-02 {2.63×10^{-2}}

Char:

This is a keyword used to indicate the character type data. A character in C language is defined as the single character enclosed within a pair of apostrophes.

eg 'a', 'p', '2', '?'

Double:

This is a keyword used to indicate a double precision floating number. Normally it is equivalent to float, but the number of digits stored after decimal are 16 in double as compared to 6 in float.

eg 234.00000000000000

Data Type Modifier

The basic data types can be modified using a series of type modifiers to fit the requirements of a particular program.

Signed

This can be applied to integer variables. The default declaration assumes a signed number. The range of signed integer lies between -32768 to 32767

Unsigned

It can be used for integer, It can be combined with long or short. If the keyword unsigned is used before integer the range of positive integers are doubled & negative integers are removed.

Range is 0 to 65535

Long: This is also applied to int data types when applied to int, it essentially doubles the length in bits of data type that it modifies. It can also be used with double.

Long int ranges from -2147483648 to 2147483647

Short: This makes the size of an int half. Default declaration of int is short.

Range is same as that of integers in C language

Range is -32768 to +32767

Variable Declaration

⇒ If the variable is declared as const variable, its value cannot be changed. An attempt to modify its value will result in an error message.

eg: const float PI = 3.142;

⇒ If the variable is defined as volatile variable or just as variable then its value can be modified at any time.

eg: volatile int x;

⇒ multiple assignment to the variables is possible in C language, It is possible to assign a value to multiple variables at same time (called as multiple assignment)

eg: int a = b = c = 20;
float x = y = z = 3.687;

Backslash Constant

→ A backslash constant is a combination of two characters in which the first character is always a backslash (\), and the second character can be any one of characters. a, b, n, t etc.

→ The backslash character constant are also called escape sequence.

→ The backslash constants are used in output statements.

eg

"\a"	System bell
"\b"	back space
"\n"	new line
"\t"	horizontal tab
"\v"	vertical tab
"\0"	null (end of string)

Symbolic Constants :

A symbolic constant is a name that substitute a numeric constant, a character constant or string constant.

eg

```
area = 3.142 * radius * radius;
```

here it can be defined as

```
# define PI 3.142
```

now. the formula is modified as

```
area = PI * radius * radius;
```

⇒ Generally, the symbolic constants are written in upper case letter.

⇒ Symbolic constant must be defined at the beginning of program. ie before main().

Storage Class :

The storage class determines the part of the memory where the variable would be stored & how long the storage allocation continues to exist ie how long the variable would exist.

It also determines the scope of the variable which specifies the part of program where the variable name is visible.

Storage class also determines the initial value of the variable, if the variable is not explicitly initialized by any value.

① Automatic Storage Class

→ Variables declared at the start of a block are automatic variables

→ memory is allocated automatically upon entry to the block & freed automatically upon exiting the block

→ Variables are accessible only within the block, hence the scope of automatic variables are local to the block in which they are defined.

→ No block outside the defining block may have direct access to automatic variables.

→ By default all variables declared within a function are automatic

They can be declared as:

```
int i;
auto int i;
```

→ To declare any variable as automatic, a keyword auto is prefixed before its declaration.

No Need

eg

```
# include <stdio.h>
void main ( )
{
  autoblock ( );
  autoblock ( );
}

auto block ( )
{
  auto int x = 10;
  printf ("\nvalue of x is %d", x);
  x = x+10;
}
```

Output is

Value of x is 10
Value of x is 10.

Note:

(i) Storage : Memory

(ii) Default initial value : Garbage value.

(iii) Scope : Local to block in which it is declared

(iv) Life : As long as program control remains within the block in which it is declared.

eg

```
#include <stdio.h>
void main()
{
auto int i, j;
printf("%d %d\n", i, j);
}
```

Output

1211 221 {garbage value}

eg

```
# include <stdio.h>
void main( )
    {
      auto int i= 1;
      {
        auto int i=2;
        {
          auto int i=3;
          printf ("%d", i);
        }
        printf ("%d", i);
      }
      printf ("%d", i)
    }
```

Output:

3 2 1

Register Storage Class

→ The register storage class specifier indicates to the compiler that the value of variable should reside in computer system register.

→ As the size of computer memory is limited, it is possible to keep only few variables, if compiler is not able to put variables in computer register, then the

variable are treated as automatic variables.

→ Thus it is not sure that the variable is stored in computer register.

→ Frequently used variables like loop counter are stored in registers.

Note:

Storage : CPU registers

Default Initial value: Garbage Value

Scope: local to block in which variable is declared

Life: Till control remains within the block in which the variable is defined

eg

```
# include <stdio.h>
void main()
{
  register int i;
  for (i=1; i<=10; i++)
    printf("%d", i);
}
```

Static Storage Class

→ The static storage class provides a lifetime to the variables over the entire program, but it restricts the scope of static variable.

→ These variables are declared using the static keyword.

→ Default initial value of static variables is zero

→ Static variables retain their values even after the block in which they are defined terminates.

→ The value of a static variable in a function is retained between repeated call to the same function.

Note

Storage : Memory
Default initial value : Zero
Scope : Local to the block in which the variable is declared
Lifetime : Variables retains its value between different function calls.

eg

```
# include <stdio.h>
void main ( )
    {
      increment ( );
      increment ( );
      increment ( );
    }

increment ( )
    {
      static int i = 1
      printf ("%d \n", i);
        i = i + 1;
    }
```

Output

In case of (auto int).

1
1
1

In case of (static int)

1
2
3

External Storage Class :

→ External storage class variables are accessible through out the program execution ie all the functions / blocks use that particular variable.

→ These variables are known as global variables.

→ External variables are declared outside any functional block

→ Memory is allocated to these variables when program begins execution & the lifetime is untill the program terminates

→ Default initial value of these variables is zero.

→ Scope of these variables is global ie entire program code has accessibility to these variables

→ All functions may access external variable by its name, but if any variable declared within the function have same name, reference to the name access the local variables.

Note:

Storage: Memory
Default initial value: Zero
Scope : In all functions of program (global)
Lifetime: Throughout program execution

eg

```
#include <stdio.h>
int i;
void main()
  { printf ("\n i= %d", i);
    increment ( );
    increment ( );
    decrement ( );
    decrement ( );
  }
increment ( )
    { i= i+1;
    } printf ("\n on incremental i= %d", i);
decrement ( )
      { i= i-1;
      } printf ("\n on decrement i= %d", i);
```

output

```
i=0
on incremental i= 1
on incremental i=2
on decrement i= 1
```

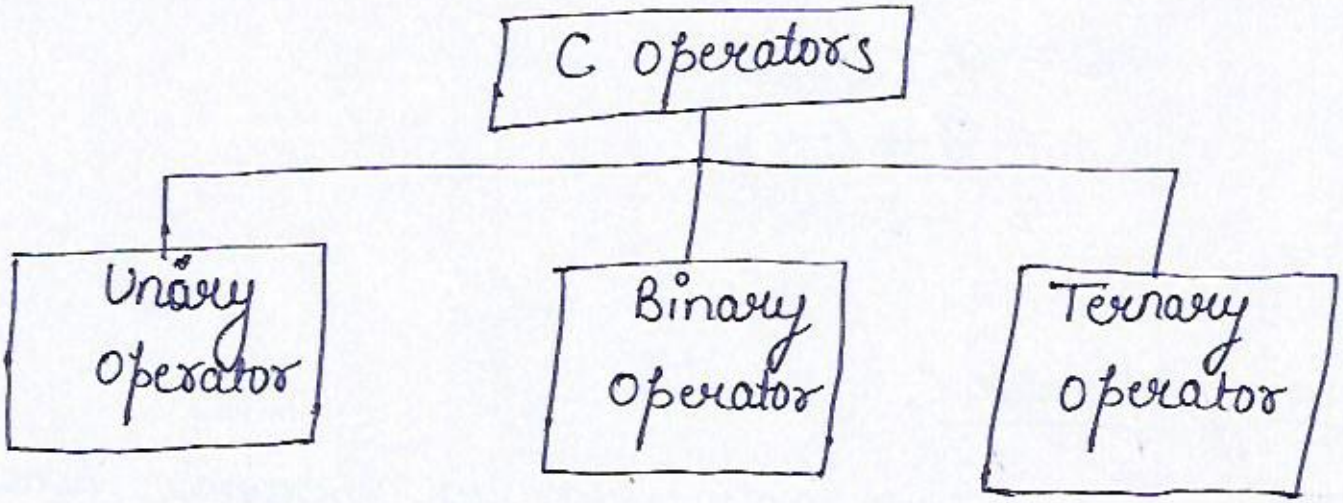
Operators

→ In order to carry out basic arithmatic operations & logical operations operators are needed.

→ Operators act as connectors and they indicate what type of operation is being carried out.

→ The values that can be operated by these operator are called operands.

→ operators in C language are broadly classified in three categories.



Unary Operator

An operator that acts upon only one operand is known as unary operator. Some unary operators are:

① Unary minus
② logical NOT operator (discussed in logical operator)
③ Bitwise complementation (discussed in Bitwise operator)

Unary Minus:

Any positive operand associated with unary minus operator gets its value changed to negative.

eg $a = 3, \quad b = 4$

$$c = a + (-b)$$
$$= 3 + (-4).$$
$$= -1$$

Binary Operator:

These operators acts upon two operands, hence named as binary. The binary operator are further divided in four categories

① Arithmatic operator
② Logical operator
③ Relational Operator
④ Bitwise operator

Arithmatic Operator:

Arithmatic operators peryorms basic arithmetic operations such as addition, substraction, multiplication & division.

Modulus operator is also used in C language that finds the remainder after integer division.

Operation	Operator	Precedence	Associativity
Addition	+	2	Left to Right
Substraction	−	2	Left to Right
multiplication	*	1	Left to Right
Division	/	1	Left to Right
Modulus	%	1	Left to Right

* The precedence indicates which terms to be evaluated.
* Associativity indicates the order in which the terms are evaluated.

Evaluation of Arithmatic Expression :

(i) $a+b$

⇒ There is only one operator (+), hence the value of 'a' is added to value of 'b'.

(ii) $x*y/z$

⇒ There are two operators ('*' & '/') having same priority. Now expression is scanned from left to right.

(I^st) $x*y$ is evaluated

(II^nd) The result of $(x*y)$ is divided by z,

(iii) $a+b*c/d-e$

(I^st) $b*c$

(II^nd) $(b*c)/d$

(III^rd) $a+((b*c)/d)$

(IV^th) $(a+((b*c)/d))-e$

eg:

```
int n1, n2;
n1 = 5;
n2 = 3;
```

(i) $n_1 + n_2 = 5+3 = 8$

(ii) $n_1 - n_2 = 5-3 = 2$

(iii) $n_1 * n_2 = 5*3 = 15$

(iv) $n_1 / n_2 = 5/3 = 1$

(v) $n_1 \% n_2 = 5\%3 = 2$

eg

```
float a, b;
a = 5.0
b = 2.0
```

(i) $a+b = 5.0+2.0 = 7.0$

(ii) $a-b = 5.0-2.0 = 3.0$

(iii) $a*b = 5.0*2.0 = 10.0$

(iv) $a/b = 5.0/2.0 = 2.5$

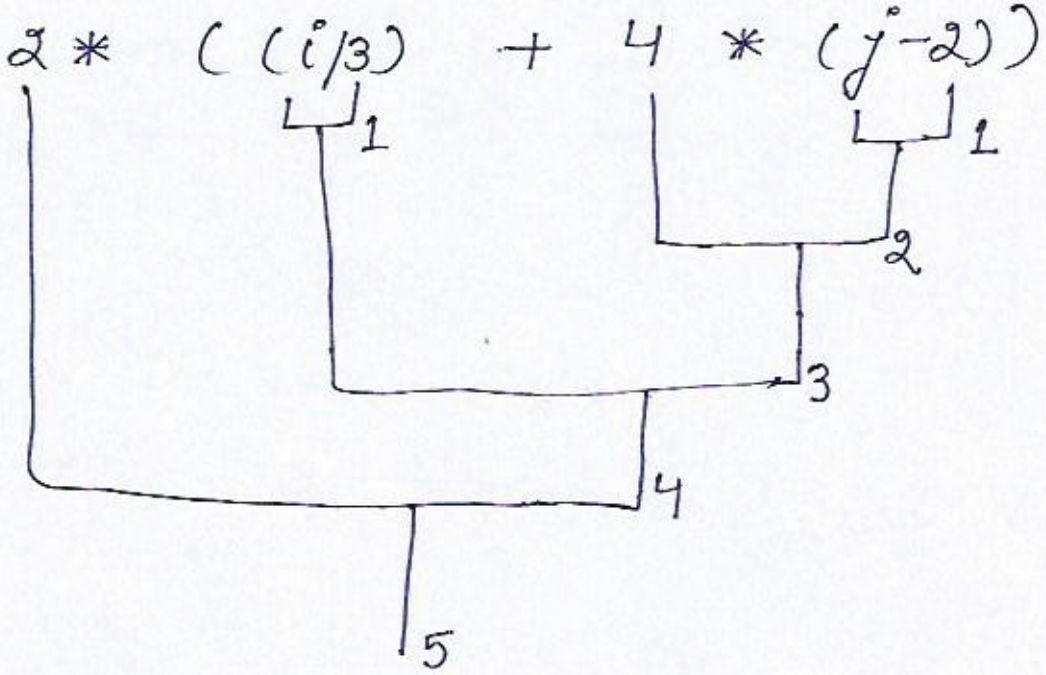
Note: % operator cannot be used with floating values

Rules for Evaluation of Arithmetic Expressions

① If the expression have paranthesses, then the expression inside parentheses must be evaluated first. (Rest priority & associativity will remain same)

② If unary minus is present in the expression then the term associated with unary minus must be evaluated before other expressions.

eg $2 * ((i/3) + 4 * (j-2))$ where $i=8$ & $j=5$



Thus

$2 * ((8/3) + 4 * (5-2)) = 2 * (2 + 4 \times 3)$

$= 2 * (2 + 12)$

$= 2 * 14$

$= \boxed{28}$

Program to illustrates Basic Arithmetic operators


```
# include <stdio.h>
# include <conio.h>
 void main ( )
      {
        int a, b;
      int sum, dif, prod, quot, rem;
     printf (" Enter value of first number");
     scanf ("%d", &a);
     printf (" Enter value of second number");
      scanf ("%d", &b);

     sum = a+b;
     dif = a-b;
     prod = a*b;
     quot = a/b;
     rem = a%b;
    printf ("\n Sum = %d", sum);
     printf ("\n Difference=%d", dif);
     printf ("\n PRODUCT = %d", prod);
     printf ("\n Quotient = %d", quot);
     printf ("\n Remainder=%d", rem);
}
```

Increment & Decrement Operator

* Increment operator is used to increment the value of an integer quantity by one.

→ This is represented by '++' symbol, This symbol can be placed before or after the integer variable.

ie

```
int a;
a++ or ++a;   {both are allowed}
```

* Decrement operator is used to reduce the value of an integer quantity by one.

→ This is represented by '--' symbol. This symbol can be placed before or after integer variable.

ie

```
int b
b--  or  --b  {both are allowed}
```

Difference between Pre & Post increment/decrement

++a → indicates pre increment in the value of variable before it is used

a++ → indicates post increment in the value of variable after it is used

eg

```
# include <stdio.h>
void main ( )
{
    int a=3, b=6;
    printf ("a = %d \n", a++);
    printf ("b = %d \n", ++b);
}
```

Output

a = 3
b = 7

Similarly

--b → decrement the value of variable first then use it

b-- → use the value of variable then decrement it

```
#include <stdio.h>
void main()
{
    int m=6, n=6;
    printf("m=%d", --m);
    printf("\n n=%d", n--);
}
```

output

```
m = 5
n = 8
```

Relational operators

Relational operators are used to compare two operands, They also define the relationship existing between two constants or variables.

→ They result in either a TRUE or FALSE value. The value of TRUE is nonzero (ie 1) and the value of FALSE is Zero

→ There are six relational operators in C language.

Operator	Meaning	Precedence	Associativity
<	is less than	1	Left to Right
<=	lesser than or equal to	1	Left to Right
>	greater than	1	Left to Right
>=	greater than or equal	1	Left to Right
==	is equal to	2	Left to Right
!=	not equal to	2	Left to Right

eg

```
# include <stdio.h>
void main ()
  {
    int a = 8;
    int b = -5;
printf("%d < %d is %d \n", a, b, a<b);
printf("%d == %d is %d \n", a, b, a==b);
printf("%d != %d is %d \n", a, b, a!=b);
printf("%d > %d is %d \n", a, b, a>b);
printf("%d <= %d is %d \n", a, b, a<=b);
printf("%d >= %d is %d \n", a, b, a>=b);
}
```

Output

8 < -5 is 0
8 == -5 is 0
8 != -5 is 1
8 > -5 is 1
8 <= -5 is 0
8 >= -5 is 1

Logical Operator

Logical operators are used to take decisions, There are three such operators, (AND, OR & NOT).

→ AND, OR are binary operator & NOT is unary operator

→ The result of these operators is either TRUE (1) or FALSE (0).

→ Logical operator are used to connect one or more relational expression.

Operator	Meaning	Precedence	Associativity
&& (double ampersand)	Logical AND	2	Left to Right
|| (double pipeline)	logical OR	3	Left to Right
! (exclamation)	logical NOT	1	~~Left to Right~~ Right to Left

AND

Operand 1	Operand 2	operand 1 && operand2
0	0	0
0	1	0
1	0	0
1	1	1

OR

Operand 1	Operand2	operand 1 ~~&&~~ || Operand2
0	0	0
0	1	1
1	0	1
1	1	1

NOT

Operand	! Operand
0	1
1	0

eg Suppose

$a=2,\ b=4,\ c=3$

a && b || c && (!b)
2 && 4 || 3 && (!4)
2 && 4 || 3 && 0
1 || 3 && 0
1 || 0
= 1

Prog

```
# include <stdio.h>
void main()
{
    int a=4, b=3, c;
    c = a && b;
    b = a || b || c;
    a = a && b || c;
    printf("%d.%d %d", a, b, c);
}
```

Output

1, 1, 1

Bitwise Operator :

All the data stored in the computer's memory as a sequence of bits (ie 0, 1), some application needs manipulation of these bits.

→ To perform bitwise operations, C provides six operators.

→ These operators works on Char & Int variables but not on floating type data.

~~Theta~~ .

Operator	Symbol Name	Meaning
&	Ampersand	Bitwise AND
|	Pipeline	Bitwise OR
^	Caret	Bitwise XOR
~	Tide	1's complement (NOT)
<<	Double less than	Left shift of bits
>>	Double greater than	Right shift of bits

XOR

operand1	operand2	operand1 ^ operand2
0	0	0
0	1	1
1	0	1
1	1	0

eg int a=4, b=3;

For 8 bit Computer

Binary Equivalent of 4 & 3 are

a= 0000 0100

b= 0000 0011

now

a & b = 0000 0000

a | b = 0000 0111

a ^ b = 0000 0111

Bitwise Complement :

let a=10

then a = 1010

if b = ~a = ~(1010)

then b = 0101

Bitwise left shifting

In left or right shift operator case, the general form of representation is

<variable>	<shift operator>	<nob>

variable → int or char data element

shift operator → left shift or right shift operator

nob → number of bits to shift

eg: (i) a >> 1 or (ii) a << 2

left shift

Let a = 132

or 10000100

Before left shifting

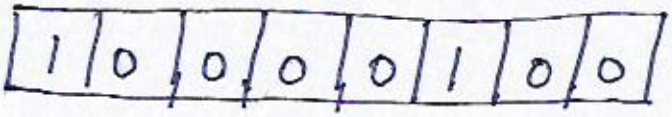
1	0	0	0	0	1	0	0

After one bit shifted to left

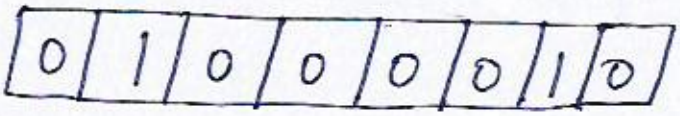
0	0	0	0	1	0	0	0

Right Shift

Before shifting



After one bit Right shifting



Program

```
# include < stdio.h >
Void main ( )
    {
     int x, y;
     x=128, y= 32;
   x = x >> 1;
printf ("After right shifting by 1, x= %d \n", x);
   y = y<<2;
printf ("After Left shifting by 2, y= %d \n", y);
}
```

Output

After right shifting by 1, x = 64

After left shifting by 2, y = 128

Ternary Operator

* Ternary operator is conditional operator in C.
* It takes three operands.
* '?' is used as a ternary operator in C
* The general form of a ternary expression is as follows:

<expression> ? <value1> : <value2>

where,

expression → Relational expression
value1 → value to be assigned on true
value2 → value to be assigned when the result of expression is false

eg: c = a<b ? a : b;

here c is assigned a value of a if a is less than b. Otherwise it will be assigned the value of b.

Prog

```
# include < stdio.h>
void main ()
{
    int a=4, b=5, result1, result2;
    result1 = a>b ? a : b;
printf ("The result1 = %d \n", result1);
```

```
result2 = a<b? a: b;
    printf ("The result 2= %d\n", result2);
}
```

Output :

The result 1 = 5

The result 2 = 4

Special Operator of C

Comma Operator: The comma operator is basically associated with for statement. It is also used to link two or more related expressions together. In such case the expression are evaluated in left to right fashion and we can obtain the value of the combined expression ~~from the value of the combined expression~~ from the value of rightmost expression.

eg. Sum = (a=12, b=22, a+b).

here there are 3 expressions, the value of last expression is set to sum.

Size of ( ) operator

→ The size of() operator returns the size(in number of bytes) of the operand.

→ It is written in lower case letters.

→ It must precede its operand.

→ The operand may be constant, variable or data type.

eg

```
x = size of (int);
y = size of (sum);
```

Program

```
void main()
{ int x = 10;
  float y = 6.5;
  char ch = 'y';
printf("size of x = %d\n", size of (x));
printf("size of y = %d\n", size of (y));
printf("size of ch = %d\n", size of (ch));
printf("size of double = %d\n", size of (double));
}
```

Output

size of x = 2

size of y = 4

size of ch = 1

size of double = 8

Precedence & Associativity of all C operators

Operator Category	Operator	Precedence	Associativity
Parentheses, braces	(), []	1	Left to Right
Unary operator	+, -, ++, --, !, ~, &	2	Right to Left
multiplicative operator	*, /, %	3	left to Right
Additive Operators	+, -	4	Left to Right
Shift operator	<<, >>	5	left to Right
Relational operator	<, <=, >, >=	6	left to Right
Equality operators	==, !=	7	left to Right
Bitwise operators	&, ^, |	8	left to Right
Logical operators	&&, ||	9	left to Right
Conditional operator	?, :	10	Right to left
Assignment operator	=, +=, -=, *=	11	Right to left

Assignment Operator:

→ Assignment operator assigns the value of expression on the right side of it to the variable on the left of it.

→ The assignment expression evaluates the operand on the right side of operator (=) & places its value in the variable on the left.

eg
```
a=4;
area= length * breadth;
```

C language also provides a compact way of writing assignment statements in expressions.

The general form of shorthand assignment in C is as follows:

Var	operator	=	expression	;

Var → Variable

operator → an arithmetic or bitwise operator

expression → is simply an expression.

Operator	Assignment (a & b)	Shorthand assignment
+	a = a + b	a += b
-	a = a - b	a -= b
*	a = a * b	a *= b
/	a = a / b	a /= b
%	a = a % b	a %= b

Data Type Conversion :

Some time it is needed to change the datatype of variable. When we change the nature of data stored in variable. This process is known as data type conversion or type casting.

Syntax (datatype) variable

⇒ eg
```
int K= 2;
float term;
term = 1/K;
```

Here the value assigned to the term is 'zero' as of integer division.

But the real value which should be 0.5 for 0.5 datatype conversion is required which can be achieved as follows:

```
term = 1/(float)K;
```

Mixed Mode Operations :

⇒ Expressions that involves, elements of type real & integer are called mixed mode expressions

→ If an expression containing both integer & float values, then the result will be of type float.

→ If the result value is assigned to integer then the fractional part of float must be truncated.

→ Operators that can be used in mixed mode expressions are (+, -, *, /, %)

eg:

```
int a=4, b=2;
float x=2, y=3
```

(i) `a = (x/y) + (a/b);`

(ii) `b = (a*x)/y;`

(iii) `x = a-b;`

Sol^n

(i)

$$a = (x/y) + (a/b)$$
$$= (2.0/3.0) + (4/2)$$
$$= 0.6 + 2$$
$$= 2.6$$

$$\boxed{a = 2}$$

(ii)

$$b = (a*x)/y$$
$$4 \times 2.0$$
$$= 8.0 / 3$$
$$= 2.66$$

$$\boxed{b = 2}$$

(iii)

$$x = a - b$$
$$= 4 - 2$$
$$= 2$$

$$\boxed{x = 2.0}$$