-Operators in C-

Programming Fundamentals

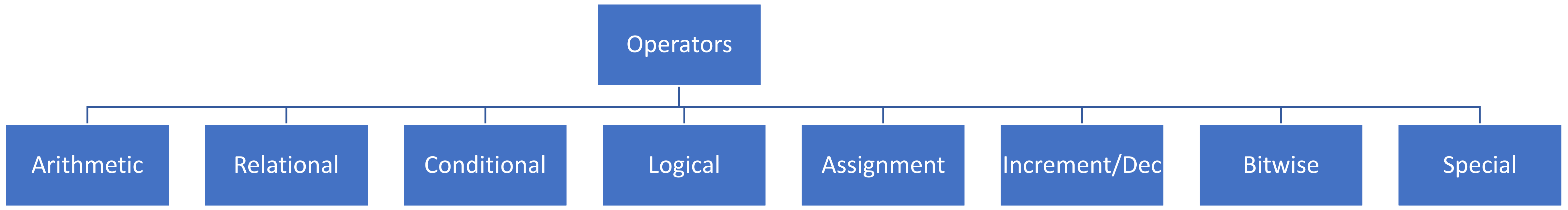
Definition: Operators

- Symbol that tells the computer to perform certain mathematical or logical manipulations.
- They are generally part of mathematical or logical expressions.

Expression: sequence of operands and operators that reduces to a single value.

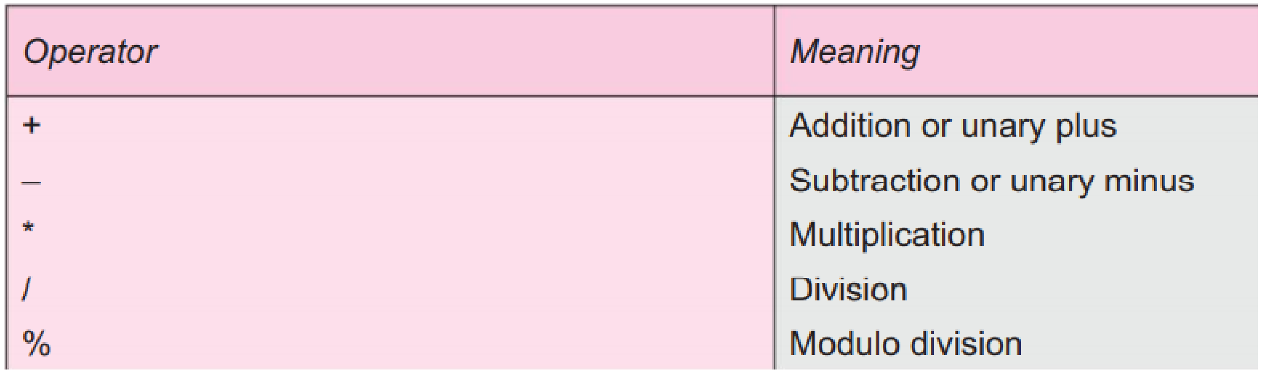
- For example, 10 + 15 is an expression whose value is 25.
- The value can be any type other than void (null /nothing)

Types of Operators



ARITHMETIC OPERATORS

- C provides all the basic arithmetic operators.
- The operators +, –,*, , % and / all work the same way as they do in other languages
- Modulo division: for remainder
- 19/5=>3 (quotient)
- 19%5=>4 (remainder)



Operator	*Meaning*
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

- These can operate on any built-in data type allowed in C.
- The unary minus operator, in effect, multiplies its single operand by –1. Therefore, a number preceded by a minus sign changes its sign.

Eg: -4: 4*-1 //unary operator

4-2: minus is binary operator here

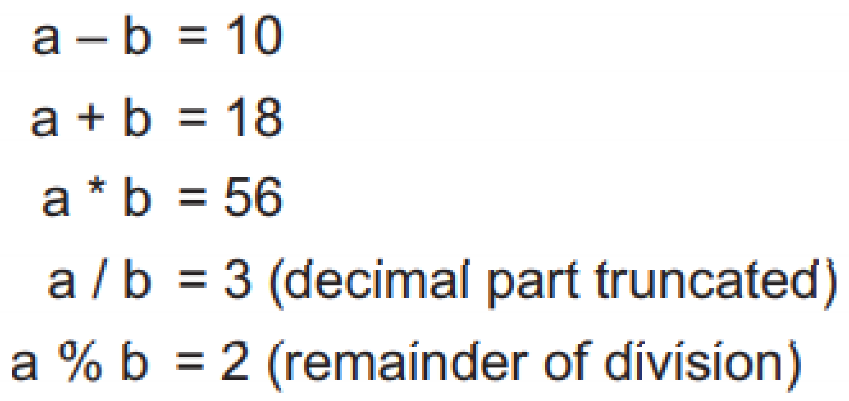
- Integer division (/) truncates any fractional part.
- The modulo division operation produces the remainder.
- The modulo division (%) cannot be used with floating point numbers.

- **Integer Arithmetic:**

When both the operands in a single arithmetic expression such as a+b are integers, the expression is called an integer expression, and the operation is called integer arithmetic.

Integer arithmetic always yields an integer value.

E.g.:



a – b = 10

a + b = 18

a * b = 56

a / b = 3 (decimal part truncated)

a % b = 2 (remainder of division)

- During integer division, if both the operands are of the same sign (+/-), the result is truncated towards zero.
- If one of them is negative, the direction of truncation is implementation dependent.

E.g.: 9/10 =>0

-9/-10 =>0

-9/10 or 9/-10 => 0 or -1 (implementation dependent)

- **Real Arithmetic**

An arithmetic operation involving only real operands is called real arithmetic.

A real operand may take values in decimal/exponent notation.

Note: The operator % cannot be used with real operands.

E.g.: $x = 6.0/7.0 = 0.857143$

$y = 1.0/3.0 = 0.333333$

$z = -2.0/3.0 = -0.666667$

- Mixed mode arithmetic:

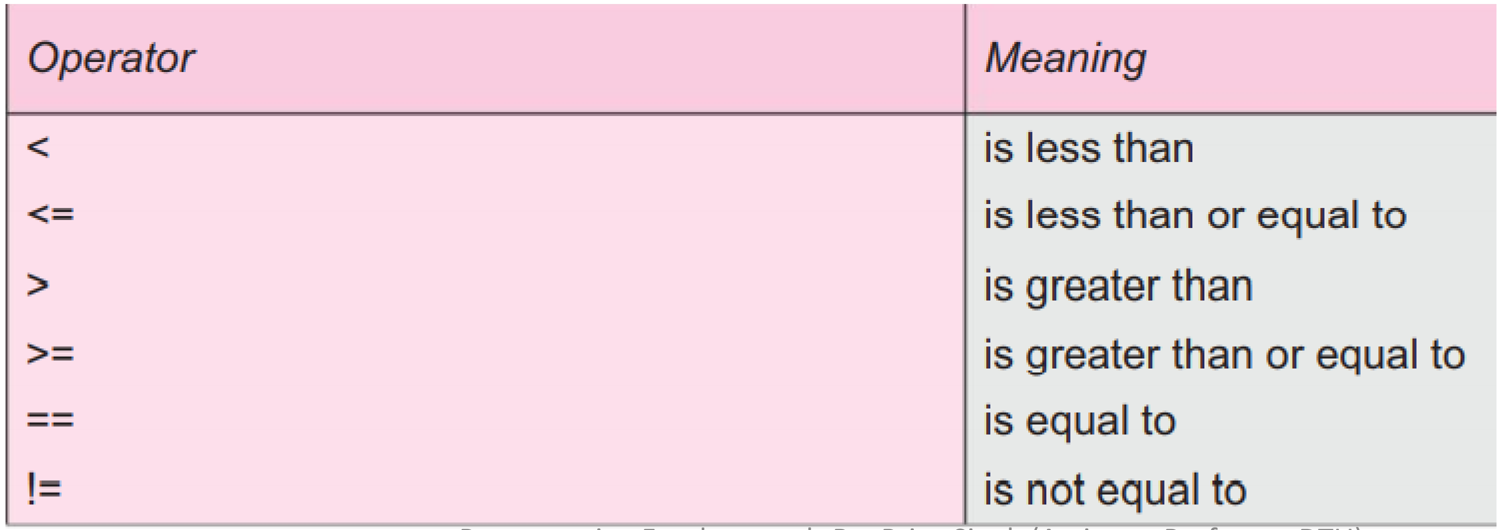
If either operand is of the real type, then only the real operation is performed and the **result is always a real number**.

Thus 15/10.0 = 1.5

whereas 15/10 = 1

RELATIONAL OPERATORS

- Used to compare two entities.
- Allows us to make decision depending on their relation.
- E.g.: age of two persons, price of two commodities etc.
- The value of a relational expression is either one(true) or zero(false).



Operator	*Meaning*
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

- A simple relational expression takes the form:

ae-1 relational operator ae-2

- (2+3/4*3) != num1, 2<=3

Where ae-1 and ae-2 can be two arithmetic expression.

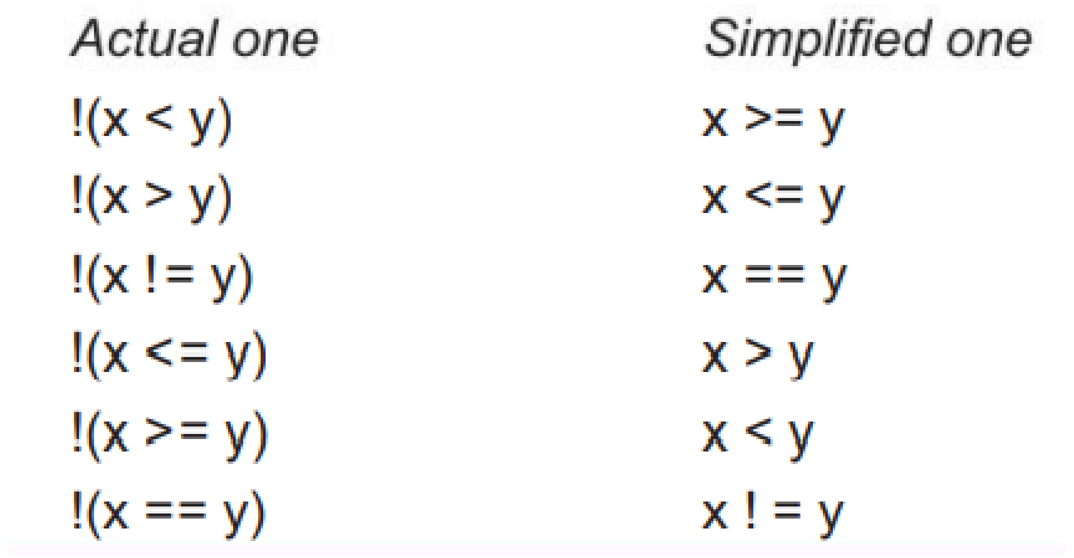
When arithmetic expressions are used on either side of a relational operator, the arithmetic expr is evaluated first.

e.g.: -1>-5, 20<1+3, var1+var2>var3+4

Among the six relational operators, each one is a complement of another operator.

>	is complement of	<=
<	is complement of	>=
==	is complement of	!=

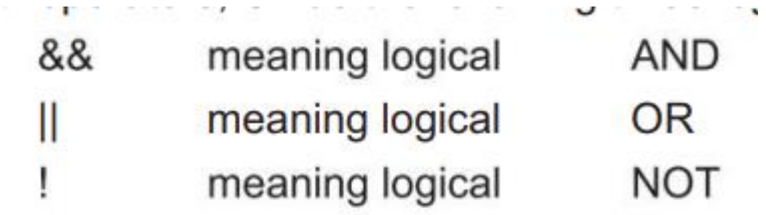
- Relational operators with negation (!) and corresponding simplified versions
- Any relational operator effect can be achieved via its complement and negation (!)



Actual one	*Simplified one*
!(x < y)	x >= y
!(x > y)	x <= y
!(x != y)	x == y
!(x <= y)	x > y
!(x >= y)	x < y
!(x == y)	x ! = y

LOGICAL OPERATORS

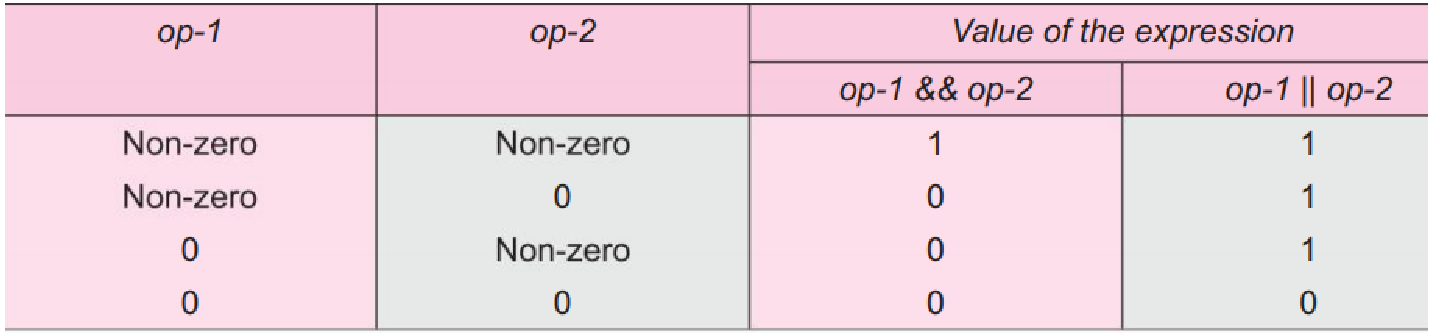
- The logical operators && (AND) and || (OR) are used when we want to test more than one condition and make decisions.



&&	meaning logical	AND
||	meaning logical	OR
!	meaning logical	NOT

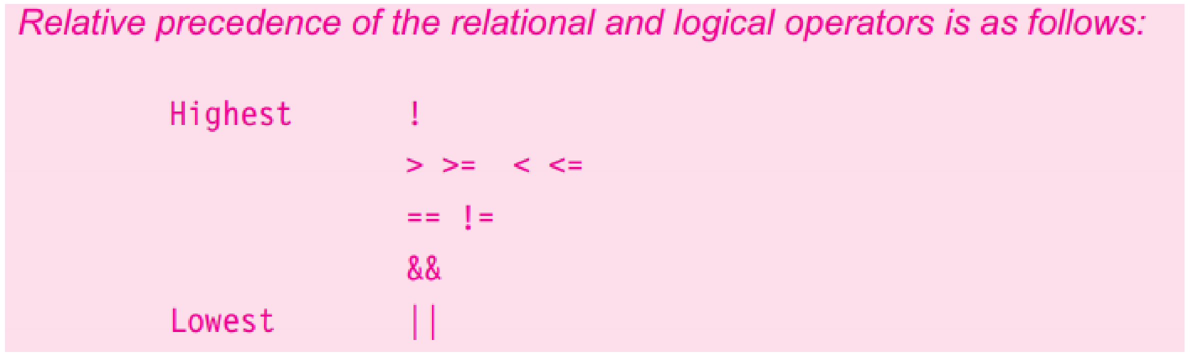
- E.g.: a>b && a!=c
- An expression of this kind, which combines two or more relational expressions, is termed as a logical expression or a **compound relational expression**.

- Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table.



op-1	*op-2*	*Value of the expression*	
		op-1 && op-2	*op-1 || op-2*
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

- Precedence of relational and logical operators: (compound expressions)



Relative precedence of the relational and logical operators is as follows:

```
Highest     !
            > >=  < <=
            == !=
            &&
Lowest      ||
```

ASSIGNMENT OPERATORS

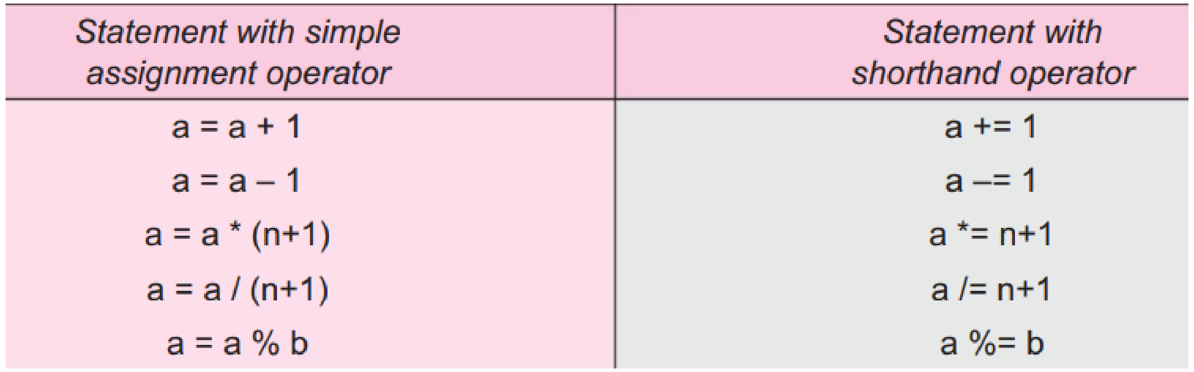
- Assignment operators are used to assign the result of an expression to a variable.
- int a=5+2; simple assignment operation
- **Compound assignment operators**: +=, -=, *=,/=
- Syntax: **v op=expression**;

Where v: variable, op=: shorthand assignment operator

e.g.:int a=2;

```
        a=a+2;
        a+=2;
        a+=b+3;
        a=a+(b+3);
```

- Some more examples:

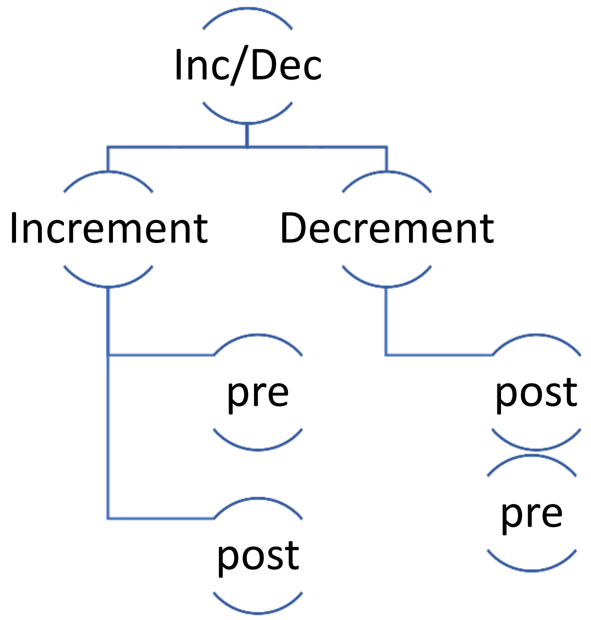


Statement with simple assignment operator	*Statement with shorthand operator*
a = a + 1	a += 1
a = a – 1	a –= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

- Advantages:

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier to read.
3. Statement is more efficient.

INCREMENT AND DECREMENT OPERATORS



- The operator ++ adds 1 to the operand, while – – subtracts 1.
- Both are unary operators (requires single operator)
- Used in for and while loops extensively

- E.g.: a++ or ++a

  b- - or - - b

  Here, ++a equivalent to a=a+1 or a+=1,

  And - -b equivalent to b=b-1 or b-=1

  While ++a and a++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.

  Solve it: m=5, n=2,j=4

  m = n++ –j+10; (2-4+10)

- Increment and decrement operators are unary operators and they require variable as their operands.
- When postfix (++/--) used with a variable in an expression, expression is evaluated first using the original/current value of the variable and then the variable is incremented (or decremented) by one.
- The precedence and associatively of ++ and – – operators are the same as those of unary + and unary –.

E.g: a++ + --b - c++

post inc, post dec (L-->R)

pre inc, pre dec (R -->L)

Execution order:

a++

c++

--b

CONDITIONAL OPERATOR

- A ternary operator in C (?) is used for conditional expression (? :)
- Syntax: ***exp1 ? exp2 : exp3***

where exp1, exp2, and exp3 are expressions.

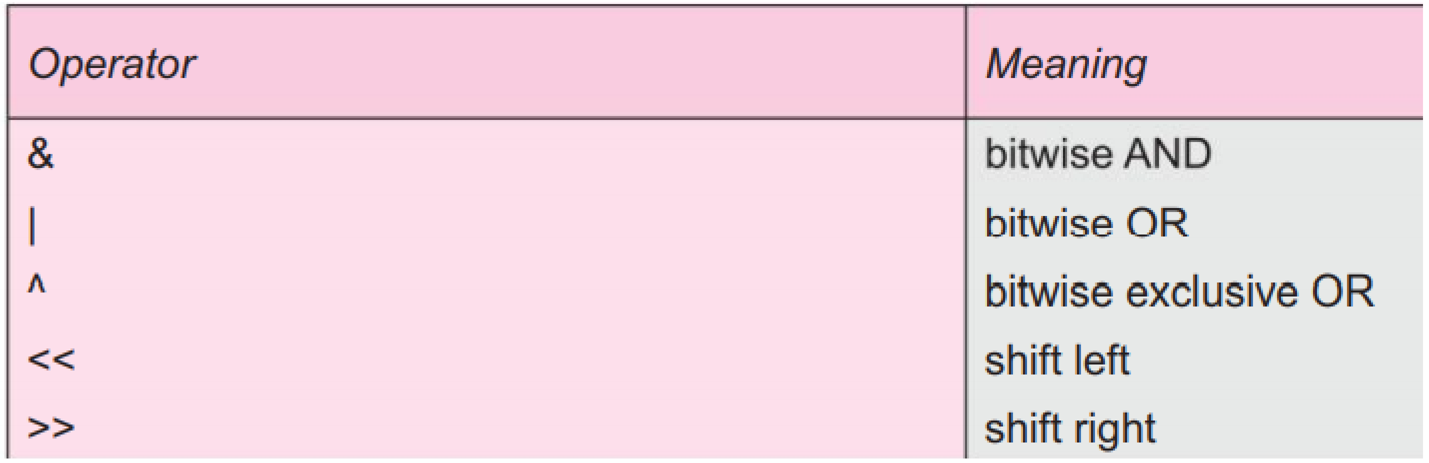
Expr1 is evaluated first (condition).
If it is true, expr2 is evaluated,
Otherwise, expr3 is evaluated.

This can be achieved using the if..else statement

```
If(expr1)
{
expr2}
else {
expr3}
```

BITWISE OPERATORS

- Used for manipulation of data at bit level.
- These operators are used for testing the bits, or shifting them right or left.
- Bitwise operators may not be applied to float or double.



Operator	*Meaning*
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right

SPECIAL OPERATORS

- C supports some special operators of interest such as comma operator, sizeof operator, pointer operators (& and *) and member selection operators (. and –>)
- Member selection operators: used with structure (derived datatype)

- Comma(,) Operator:
- Least precedence among all.

The comma operator can be used to **link the related expressions together.**

A comma-linked list of expressions are evaluated **left to right** and the value of **right-most expression** is the value of the combined expression. For example, the statement:

int **value = (x = 10, y = 5, x+y);**

First assign value 10 to x, then assigns 5 to y value and finally assigns x+y=15 to value.

Note: Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

Applications of comma (,) operator:

In **for** loops:

```
for ( n = 1, m = 10, n <=m; n++, m++)
```

In **while** loops:

```
while (c = getchar( ), c != '10')
```

Exchanging values:

```
t = x, x = y, y = t;
```

- sizeof operator:

It is a **compile time operator**. (size is compiler dependent)

When used with an operand, it returns the number of bytes.

e.g.: sizeof(variable)

 sizeof(long float)

 sizeof(2.34f)

Utility: The operator is normally used to determine the **lengths of arrays and structures** when their sizes are not known to the programmer.

It is also used to allocate memory space dynamically to variables during execution of a program.

```
int *ptr = (int*) malloc(sizeof(int)); //single memory element
int *ip = (int *) malloc( sizeof(int)*10 ); //array
```

PRECEDENCE OF ARITHMETIC OPERATORS

- An arithmetic expression without parentheses will be evaluated from **left to right** using **the rules of precedence of operators**.

- Priority levels of arithmetic operators:

1. High priority x / %
2. Low priority + - (l→r) 2-3+4x6 (resolution: x - +)

The basic evaluation procedure includes ‘two’ left-to-right passes through the expression.

1st pass: high priority

2nd pass: low priority

- E.g. 1:

$$x = a–b/3 + c*2–1$$

When a = 9, b = 12, and c = 3, the statement becomes

$$x = 9–12/3 + 3*2–1$$

and is evaluated as follows

It is evaluate as:

First pass

Step1: x = 9–4+3*2–1

Step2: x = 9–4+6–1

Second pass

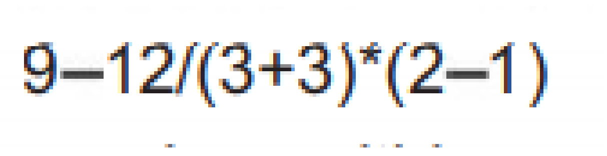
Step3: x = 5+6–1

Step4: x = 11–1

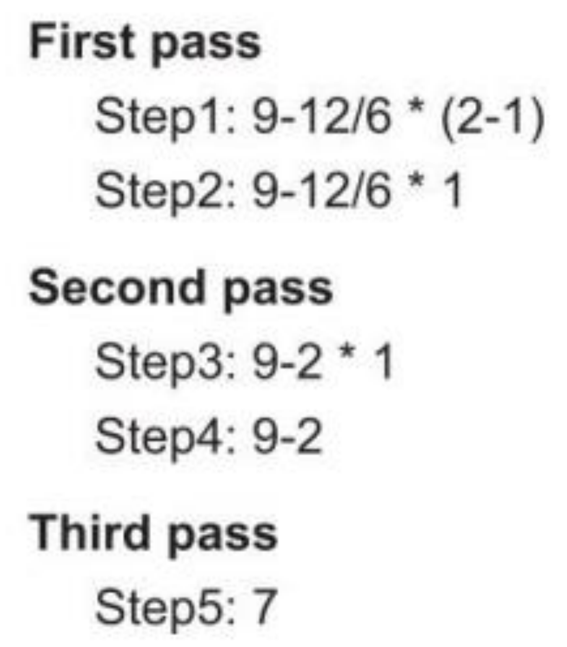
Step5: x = 10

- Order of evaluation can be changed by introducing parentheses into an expression. 2x(3+(2-3)) → Resolution (-,+, x)
- Parentheses may be nested, and in such cases, evaluation of the expression will proceed outward from the innermost set of parentheses
- E.g.:

It is eval



9–12/(3+3)*(2–1)



First pass

Step1: 9-12/6 * (2-1)

Step2: 9-12/6 * 1

Second pass

Step3: 9-2 * 1

Step4: 9-2

Third pass

Step5: 7

Rules for evaluating expressions:

- First, parenthesized sub expression from left to right are evaluated.

(2+3) / (4+3)

- If parentheses are nested, the evaluation begins with the innermost sub-expression. E.g.: (2+(3-5))
- **The precedence rule is applied in determining the order of application of operators in evaluating sub-expressions.**
- **The associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.**
- Arithmetic expressions are evaluated **from left to right** using the rules of precedence.
- When parentheses are used, **the expressions within parentheses assume highest priority.**

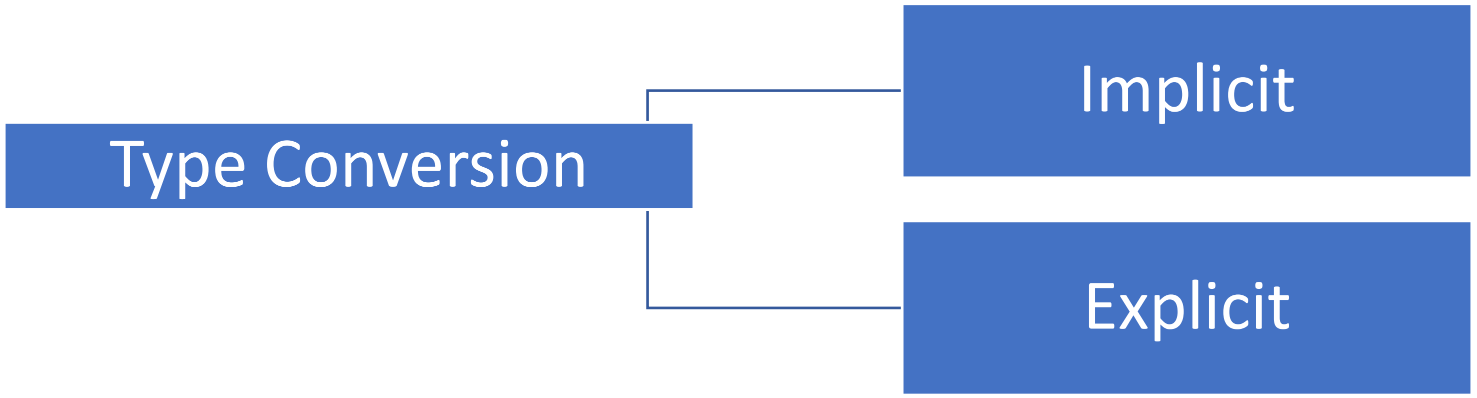
Computational Problems

- Computer gives approximate values for real numbers and the errors due to such approximations may lead to serious problems.

E.g.: a = 1.0/3; b = a * 3.0; //mixed mode arithmetic

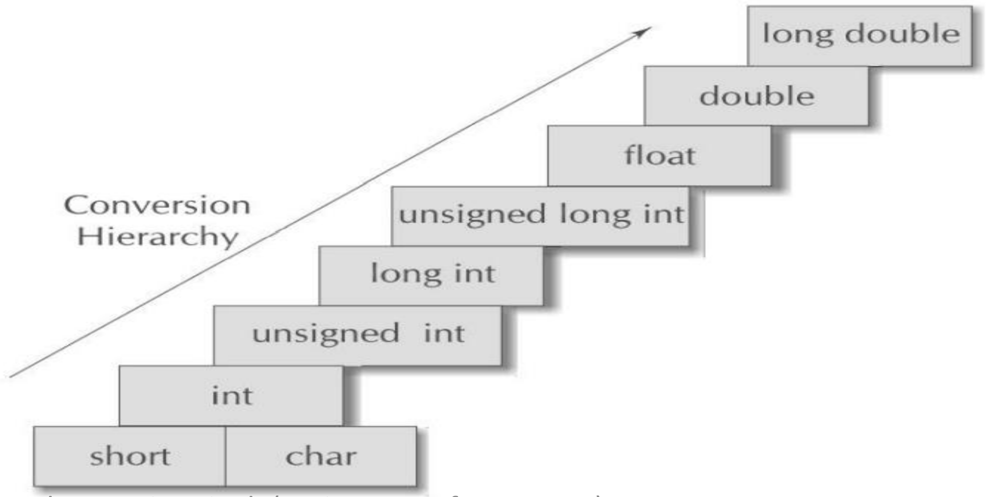
- On most computers, any attempt to divide a number by zero will result in abnormal termination of the program. Care should be taken to test the denominator that is likely to assume zero value and avoid any division by zero.
- Avoid underflow and overflow => Programmer's responsibility to handle operands with correct type and range.

TYPE CONVERSIONS IN EXPRESSIONS



a) **Implicit Type Conversion**

- C permits mixing of constants and variables of different types in an expression.
- C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance.
- During evaluation it adheres to very strict rules of type conversion.
- If the operands are of different types, the ‘lower’ type is automatically converted to the ‘higher’ type before the operation proceeds. The result is of the higher type.
- **Final result of an expression is converted to the type of variable on left side of ‘=‘ before assigning value to it**



- **Conversion Rank:**

In assignment operation , based on the hierarchy chart, operands can be

Promoted: implies that the right expression is of lower rank.

E.g.: int a=5; float b=10.9; b=b+a;

a) **Demoted**: implies that the right expression is of higher rank.

Note: Demotion may create problem.

E.. : int a=5; float b=10.9; a=b+a;

Some issues with demotion:

1. Long to int causes truncation of the fractional part.
2. double to float causes rounding of digits.
3. long int to int causes dropping of the excess higher order bits.

b) Explicit Type Conversion:

The process of a local conversion is known as explicit conversion or casting a value.

We can force a type conversion in a way that is different from the automatic conversion (as per the Conversion Hierarchy).

(type-name) *expression*

Consider, for example, the calculation of ratio of females to males in a town.

int firstNum=7, secondNum=4;

printf(“%f”,fitstNum/secondNum); →1.0

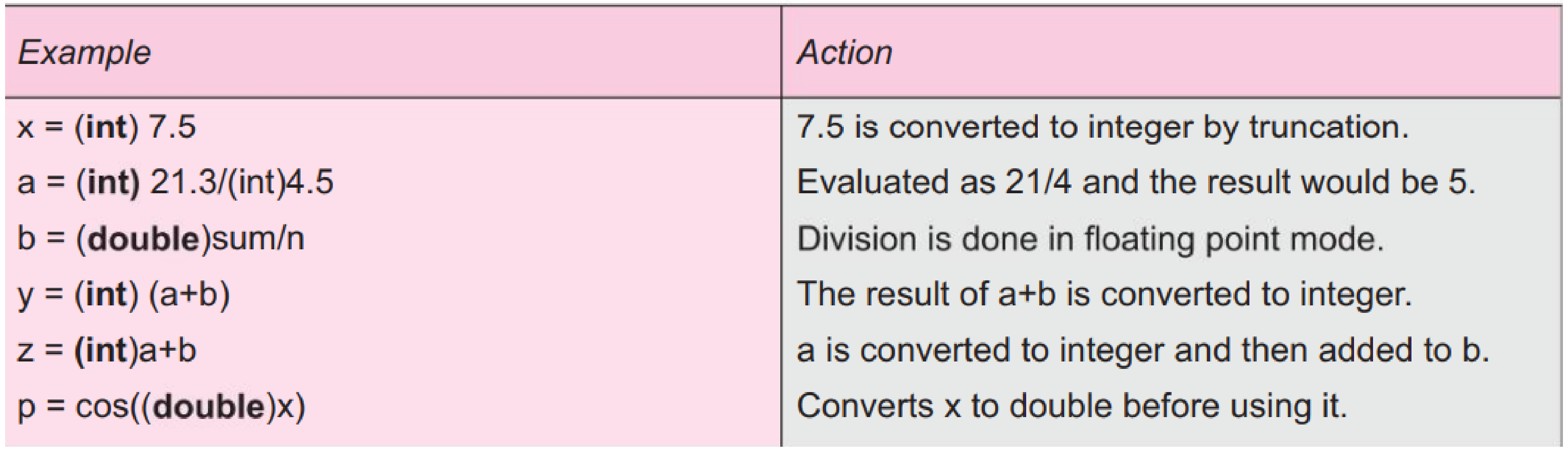
Since firstNum and secondNum are declared as integers in the program, the decimal part will be truncated.

But, we can locally convert one of the integers here to decimal (float/double; mixed-mode arithmetic) and then result will be in float therefore

printf(“%f”,**(float)**fitstNum/secondNum);

Note: Remember the effect of explicit typecasting will be applicable locally and will go off once expression is evaluated.

- **Other examples:**



Example	*Action*
x = (**int**) 7.5	7.5 is converted to integer by truncation.
a = (**int**) 21.3/(int)4.5	Evaluated as 21/4 and the result would be 5.
b = (**double**)sum/n	Division is done in floating point mode.
y = (**int**) (a+b)	The result of a+b is converted to integer.
z = (**int**)a+b	a is converted to integer and then added to b.
p = cos((**double**)x)	Converts x to double before using it.

- **Rounding-off values in an expression:**

int x,y;

y=27.6;

x = (int) (y+0.5);

Note: y+0.5 is 28.1 and on casting, the result will be type-casted to int and 28 is stored in x.

OPERATOR PRECEDENCE AND ASSOCIATIVITY

- **Operator precedence:**

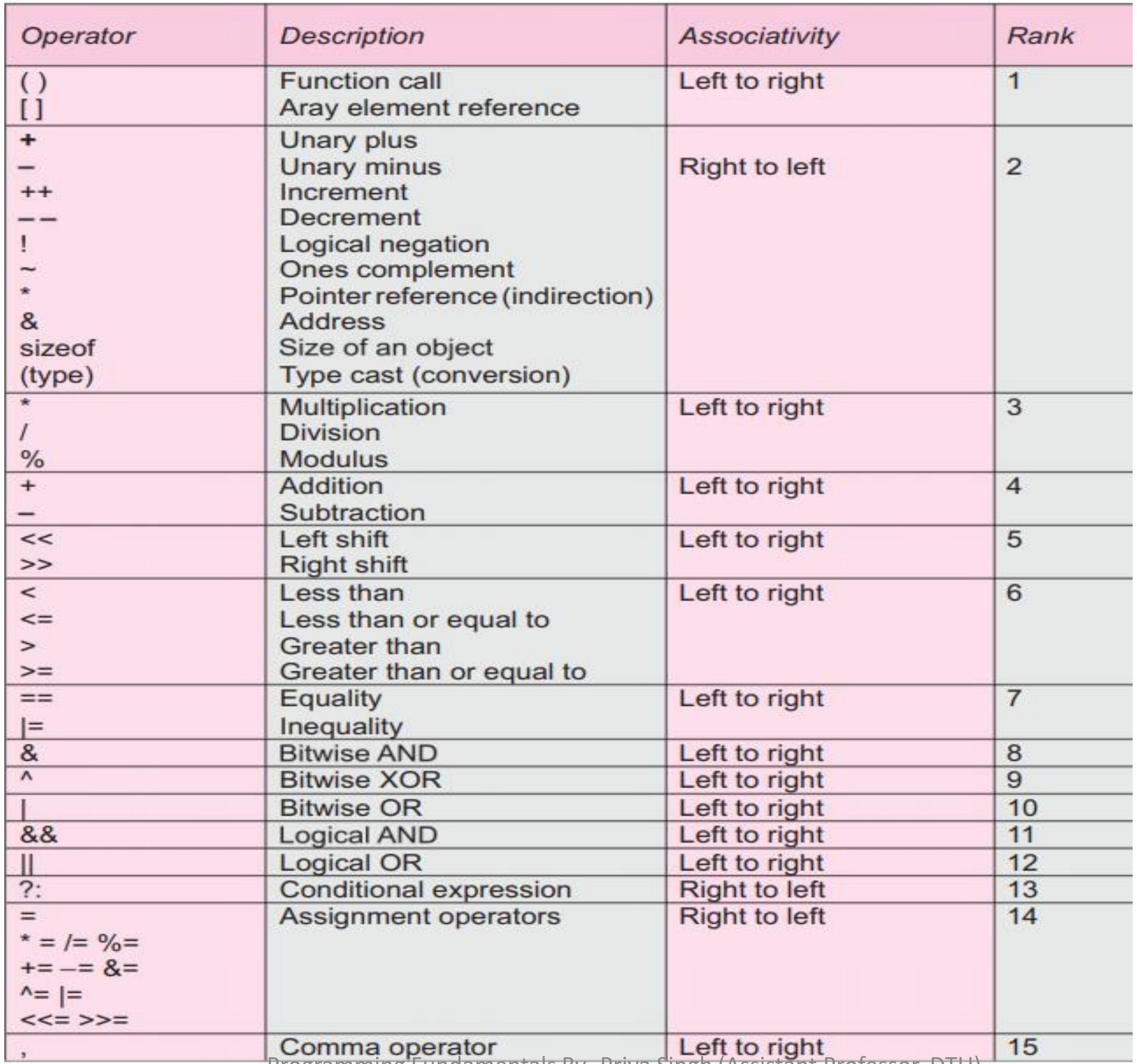
1. determines how an expression involving more than one operator is evaluated.
2. There are distinct levels of precedence and an operator may belong to one of these levels.
3. The operators at the higher level of precedence are evaluated first followed by the lower priority level.

- **Associativity of operators:**

1. Operators of same precedence are evaluated either right-left or left-right.

Note: *a) Precedence rules decides the order in which different operators are applied*

b) Associativity rule decides the order in which multiple occurrences of the same level operator are applied 2+3-4+5



Operator	*Description*	*Associativity*	*Rank*
( ) [ ]	Function call Aray element reference	Left to right	1
+ – ++ – – ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference (indirection) Address Size of an object Type cast (conversion)	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ –	Addition Subtraction	Left to right	4
<< >>	Left shift Right shift	Left to right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	6
== !=	Equality Inequality	Left to right	7
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
= *= /= %= += –= &= ^= |= <<= >>=	Assignment operators	Right to left	14
,	Comma operator	Left to right	15

Solve: `if (x == 10 + 15 && y < 10)`

Rules:

1. Priority of Arithmetic operator + is highest
2. Priority of Relational operator < is second highest
3. Priority of Equality operator == is third
4. Priority of Logical operator && is fourth

Assume a) x=12 and y=11

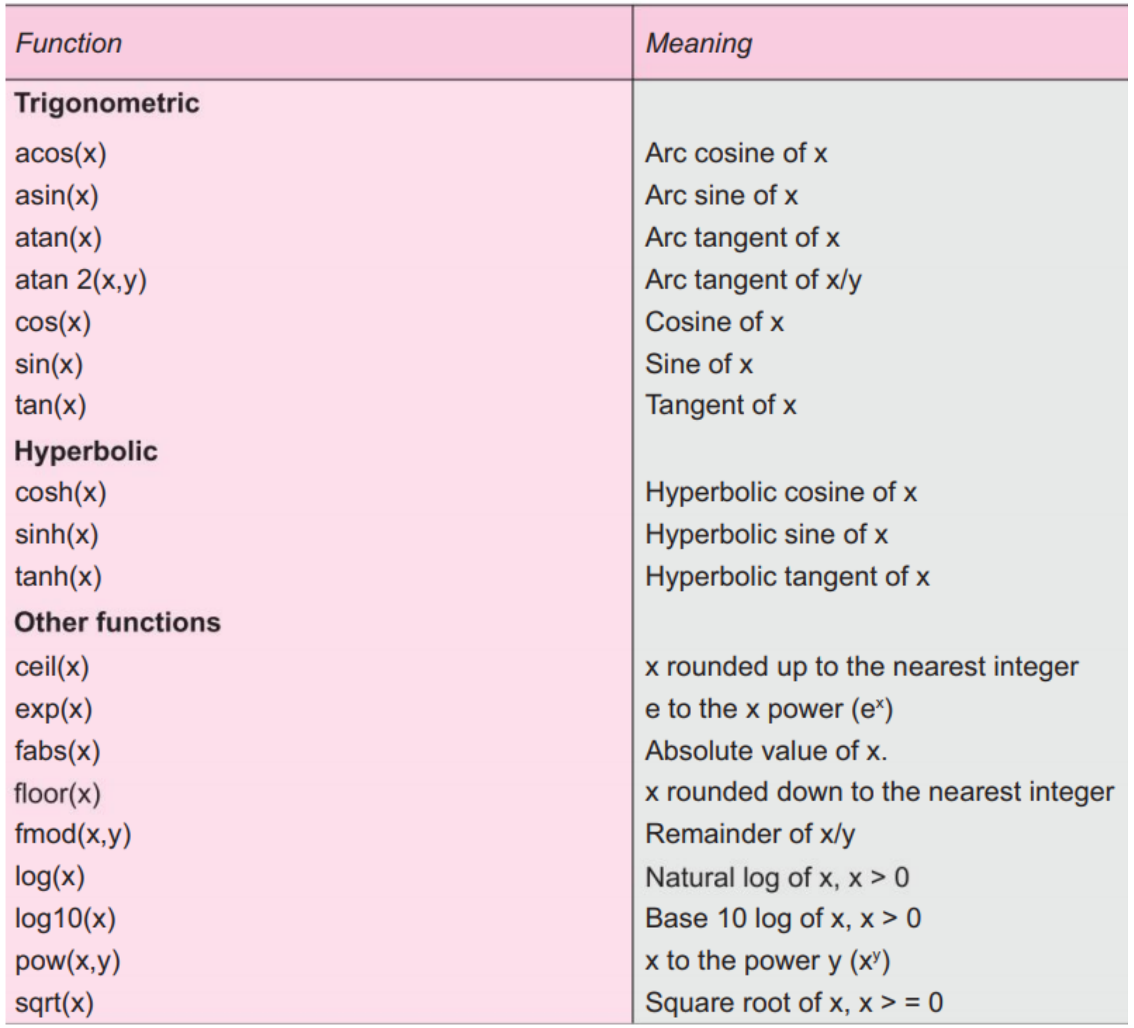
b) x=15 and y=9

Int z=10+15 && y<10

MATHEMATICAL FUNCTIONS

- Mathematical functions such as cos, sqrt, log, etc. are frequently used in analysis of real-life problems.
- Most of the C compilers support these basic math functions.

Common Mathematical Function:



Function	*Meaning*
Trigonometric	
acos(x)	Arc cosine of x
asin(x)	Arc sine of x
atan(x)	Arc tangent of x
atan 2(x,y)	Arc tangent of x/y
cos(x)	Cosine of x
sin(x)	Sine of x
tan(x)	Tangent of x
Hyperbolic	
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
Other functions	
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the x power (e^x)
fabs(x)	Absolute value of x.
floor(x)	x rounded down to the nearest integer
fmod(x,y)	Remainder of x/y
log(x)	Natural log of x, x > 0
log10(x)	Base 10 log of x, x > 0
pow(x,y)	x to the power y (x^y)
sqrt(x)	Square root of x, x > = 0

1. To use any of these functions in a program, we should include:

   #include <math.h>

1. x and y should be declared as
2. In trigonometric and hyperbolic functions, x and y are in radians.
3. All the functions return a **double**.

14.7
Ceil(14.7) → 15
Floor(14.7) →14

Log(14.7)
Pow(5,2) →25
Sqrt(25) → 5