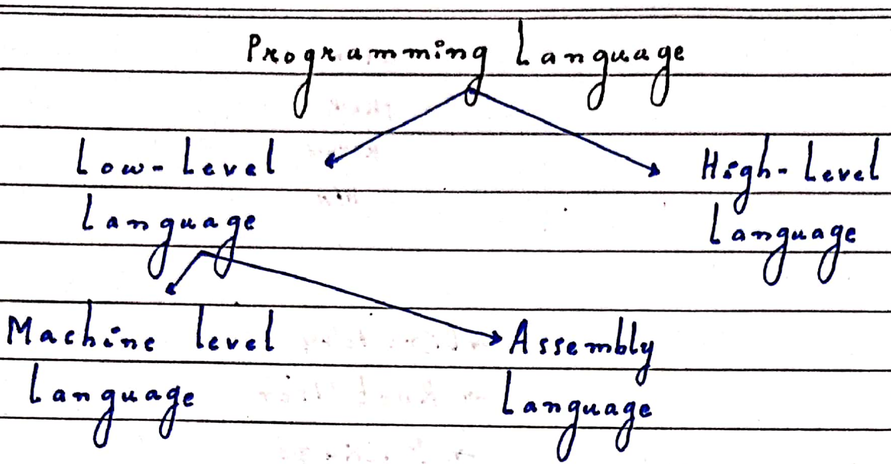


$$\text{Assembly Language} \xrightarrow{\text{Assembler}} \text{Machine Code}$$

$$\text{High Level Language} \xrightarrow[\text{Interpretor}]{\text{Compiler}} \text{Machine Code}$$

→ Machine Level language : Consists instructions that are in binary (0s and 1s).
Not easy for programmers to write instructions in binary code.

→ Assembly Language : Used english like words as ADD, SUB, MUL, etc.
Assembler translates assembly language into machine code.

→ High-level Language : Uses english like language to write statements.

Compiler and interpreter are used to translate High level language to Machine level language.

C language was developed in 1970's at Bell Laboratory by Dennis Ritchie.

Ken Thompson also developed B language.

Structure of Program

```
Comments            /* ..... */
Preprocessor Directives
Global variables
Main ()     function
{     Local variables
      statement 1;
      statement 2;
}
```

Preprocessor Director is processed by Preprocessor.

Identifiers : User defined word used to give name to entitities like variable, arrays, functions, structure, etc.

Rules for naming identifiers ;

* only alphabets (uppercase or lowercase), digits and underscore.
* First character can be alphabet or underscore.
* Keywords not allowed
* Case sensitive

for ex ; code, Code, CODE all are different.
   invalid identifiers → 5bc, int, rec#, avg no

Data

Data Types

There are 4 fundamental data types

int : to store integer value

char : to store single character

float : single precision floating point number.

double : double precission floating point number.

Type qualifier data types

→ Sign qualifier

(i) Signed

(ii) Unsigned

→ Size qualifier

(i) short

(ii) long

Char	char or signed char	1B	-128 to 127
	unsigned char	1B	0 to 255
	int or signed int	2B	-32768 to 32767
	unsigned int	2B	0 to 65535
	short int or signed short int	1B	-128 to 127
	unsigned short int	1B	0 to 255
	long int or signed long int	4B	-2147483648 to 2147483647
	unsigned long int	4B	0 to 4294967295

float	float	4B	3.4E-38 to 3.4E+38
double	double	8B	1.7E-308 to 1.7E+308
	long double	10B	3.4E-4932 to 3.4E+4932

→ Constant

1. Numeric

(i) Integer

↳ Decimal → (0-9) base 10

↳ Octal → (0-7) base 8

↳ Hexadecimal (0-9) (a-f) (A-F) base 16

(ii) Real

2. Character

Character constant : ' _ '

3. String

String Constant : " _ "

Symbolic Constant : # define MAX 100

→ Variables : is a name used to store a value, one value at a time.

→ Declaration of variable - data types

Datatype   Variable Name

int name ;   float no ;

→ Initialization of a variable ;

During declaration we can assign some value to the variable, for ex ;

```
    int   a = 10 ;
    char  ch= 'y' ;
float  x=8.9 , y=10.9 ;
int  l, m, n, total=0 ;
```

Input and Output in C

Input ⟶ Program ⟶ Output

→ We can take input from the user at runtime, using scanf().

scanf ("Control string", address 1, address 2, ...);
↳ conversion specification

→ Conversion Specification :

%c - used for character
%d - used for integer
%f - floating point number
%s - string
%o - for octal no
%x - for hexadecimal no
%g - for floating point no
%e - for floating point no
%lf - for double

```
main ()
{
    int   marks,
    scanf ("%d", &marks);
    =              ↳ ampersand
}
```

```
main ()
{
   int basic, da ;
   scanf ("%d  %d", & basic, & da);
   =
}
```

```
main ()
{
  int b;
  float f;
  char ch;
  scanf ("%d %f %c", &b, &f, &ch);
  =
}
```

```
main ()
{
  int b;
  float f;
  scanf ("%d: %f", &b, &f);
  =
}
```

```
main()
{
    int day, month, year;
    int basic;
    scanf ("%d-%d-%d", &day, &month, &year);
    scanf ("$%d", &basic);
    ≡
}
```

→ Writing Output Data

We use printf()

printf ("control string", variable 1, variable 2,...);

conversion specification and text both can be present

Example: printf("C is excellent \n");

```
2.    main ()
      {
          int   num;
          printf ("Enter the value of num");
          scanf ("%d", &num);
          =
      }

3.    main ()
      {
          int  b = 1500;
          int  h = 1000;
          int  g = 500;
      printf ("Basic = %d, HRA = %d, difference = %d", b, h, g);
        =
      }

4.    main ()
      {
        int a, b, sum = 0;
        printf ("Enter the values for a and b:     ");
        scanf ("%d %d", &a, &b)

        sum = a + b;
        printf ("The sum of a and b = %d", sum);
          =
      }
```

- Operators

1. Arithmetic Operator

Unary ← → Binary

1. Integer Arithmetic : both operands are integers.
2. Floating Point Arithmetic : both operands are of float type.
3. Mixed Mode Arithmetic : When one operand is of int type and another one is of float type.

4. Assignment Operator ('=')

Compound assignment operator

x += 5 → x = x+5

x -= 5 → x = x - 5

x*= 5 → x = x*5

x/=5 → x = x/5

x%=5 → x = x%5

5. Increment / Decrement Operator

→ Prefix Increment / Decrement Operator

++x

Prefix Increment Operator

y = ++x ;

y = x+1 ;

y = x ;

--x

Prefix Decrement Operator

y = --x ;

y = x-1 ;

y = x ;

Postfix Increment / Decrement Operator

x++

y = x++

⇓

y = x ;

~~y = x+1~~;

x--

y = x--

⇓

y = x ;

⇓

y = x-1;

Eg
```
main ()
{
    int x = 8;
    printf ("x = %d\t", x);
    printf ("x= %d\t", ++x);
    printf ("x = %d\t", x);
    printf ("x = %d \t", --x);
    printf ("x = %d \t", x++);
    printf ("x = %d \t", x--);
    printf ("x = %d \n", x);
}
```

6. Relational Operator - are used to compare values of two expression depending on thier relation,

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

Eg ; a = 9 , b = 5

a < b output = False , a != b output = true

a <= b output = False , a > b O/P = true

Logical Operator - An expression that combines two or more expression ~~For combining~~ is term as logical expression. For combining these expressions we use logical operators.

Operator	Meaning
&&	and
||	or
!	not

Conditional or Ternary Operator (? and :)

Ternary operator requires 3 expressions as operands. This is written as:

Test expression ? expression 1 : expression 2

→ If True then expr1 is evaluated

→ If False then expr 2 is evaluated

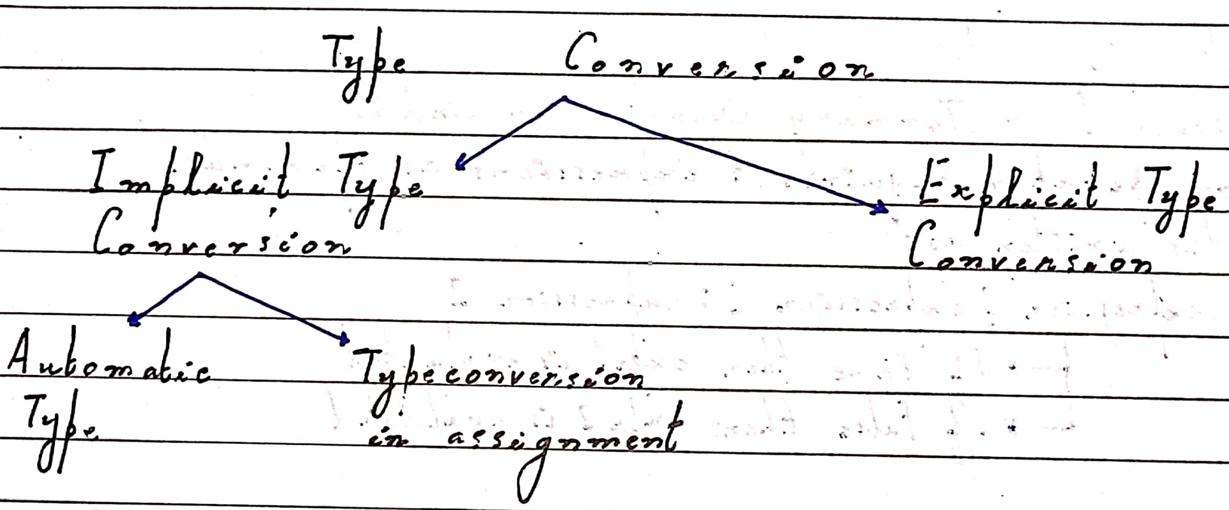
Comma Operator : is used to permit different expressions to appear in situations where only one expression will be used. The seperated expressions are evaluated from left to right and the type and value of rightmost expression is the type and value of the compound expression.

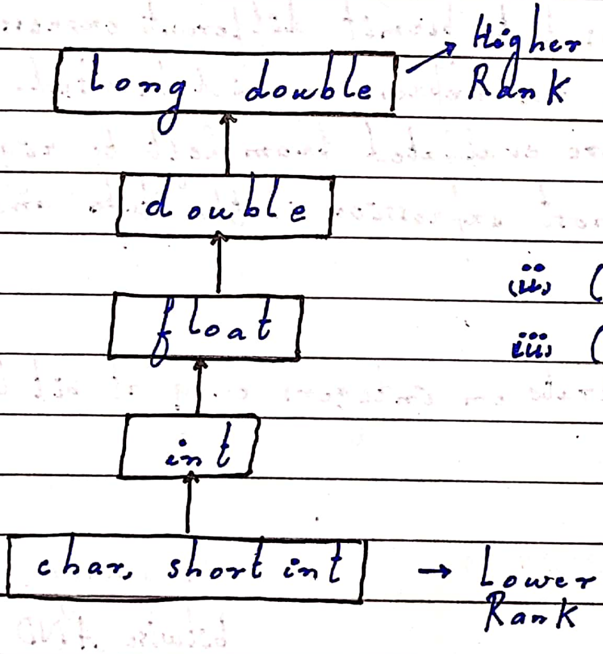
Bitwise Operators - operate on integers only at bit level. ~~bitwise operator.~~

Bitwise Operator	Meaning
&	bitwise AND
|	bitwise OR
~	one's complement
<<	bitwise leftshift
>>	bitwise rightshift
^	bitwise XOR

Type Conversion

C provides the facility of mixing different types of variables and constants in an expression. In these type of operation data type of one operand is converted into data type of other operand. This is known as type conversion.



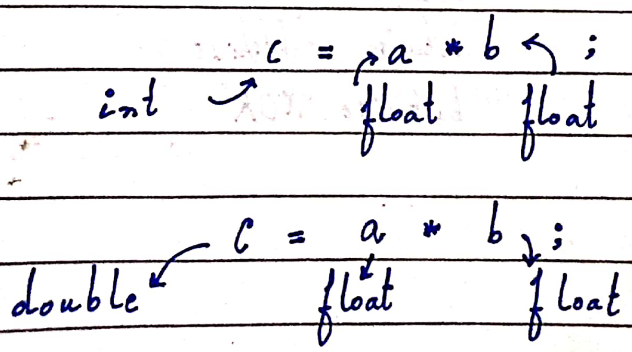


eg (i) (char, short int) * (int) → int

(ii) (int) a * b (float) → float

(iii) (float) * b (double) → double

Type conversion in assignment



R.H.S is operand is converted into L.H.S operand.

Explicit Type Conversion (Type Casting)

data type → cast operator

float z;

int a = 5 ;

int b = 2 ;

$z = \frac{a}{b}$;

2.0000;

$z = (\text{float})\left(\frac{a}{b}\right)$;

$z = \text{float}\left(\frac{20}{3}\right) = 6.00$

cast operator parathesis

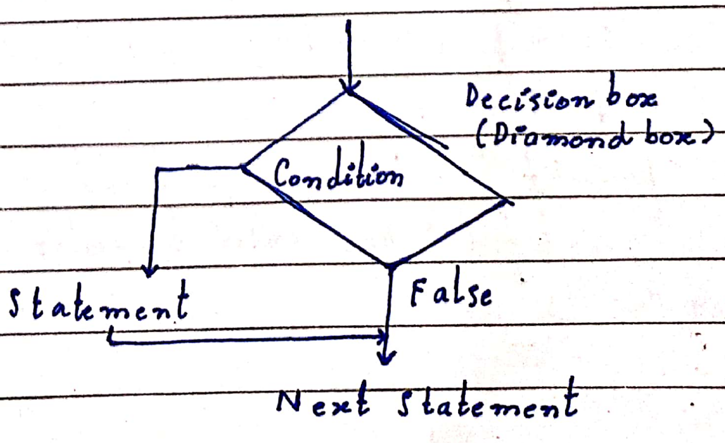
$z = (\text{float})\left(\frac{20}{3}\right) = 6.66$

→ Precedence and Associativity of Operators

operators	Precedence	Associativity
*		Left to Right
/	3	
%		
+	4	Left to Right
-		

→ Control Statements

1. if
2. if ... else
3. switch
4. loops


Decision box
(Diamond box)
Condition
Statement
False
Next Statement