17-10-2021

C - PROGRAMMING:

* Introduction :-

'BCPL' - Basic combined Programming Language
[by 'Martin Richards' at cambridge]

↓

'B Language' - first initial from BCPL.
[by 'ken Thomson' in 1970]

↓

'C Language' - second initial from BCPL.
[by 'Dennis Ritchie' in 1972 - AT&T Bell]

→ 'C' is a middle level language with 'high level' and 'low level' features.

→ 'C' is a 'structured', 'high level', 'machine independent' language.

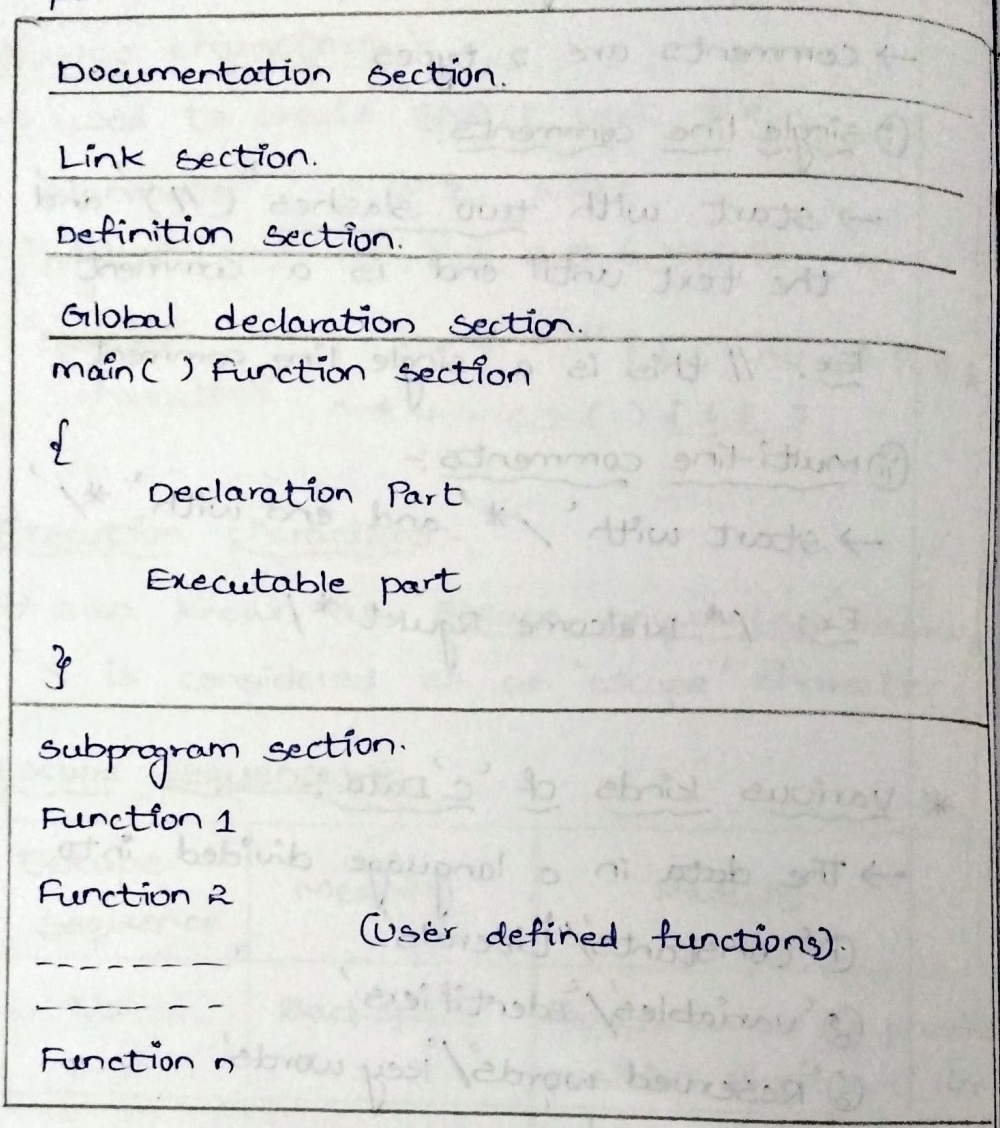
* Taxonomy of C language :-

ALGOL
↓
BCPL
↓
B
↓
Traditional C
↓
K & RC
↓
ANSI C
↓
ANSI / ISO C
↓
C95
↓
C99

* Structure of a C Program :-

→ C program is a collection of one or more functions.

→ Every function is collection of statements performs a specific task.



① Documentation section :- [optional].

→ The documentation section consists of a set of 'comment lines' giving the 'name of the program', the 'author' and other details, which the programmer would like to use later.

→ Two types of comment lines

① 'Single line comment lines' //

② 'Block Comment lines'. /* */.

→ This section is 'optional'.

② Link Section :- (essential)

→ This section provides instructions to the compiler to link functions from the system library.

Ex :- #include <stdio.h>
#include <string.h>
#include <conio.h>
#include <graphic.h>
#include <stdlib.h>
#include <mallot.h>
#include <process.h>
#include <ctype.h>

'#include' directive :-

→ It instructs compiler to include the contents of the file "stdio.h" [standard input output header file] enclosed within angular brackets.

③ Definition Section :- [optional]

→ The definition section defines all symbolic constants.

Ex :- #define PI 3.1413
#define SIZE 100
#define PRINT Print("Hi").

④ Global declaration Section :- (optional)

→ There are some variables that are used in more than one function. such are called "Global declaration & variables".

→ These are declared in global declaration section that is outside of all the functions.

→ This section also declares all the 'user-defined' functions.

⑤ main() function section :- (essential)

→ Every c program has must have one main function section.

→ This section contains 2 parts.

(i) 'Declaration part'

(ii) 'Executable part'

(i) Declaration part :- It declares 'all the variables' used in the executable part.

(ii) Executable part :- There is atleast '1 statement' in the executable part.

→ These '2 parts' must appear b/w 'opening' & 'closing braces'

→ The program execution begins at the opening brace & ends at closing brace.

→ The 'closing brace' of the main function is the 'logical end' of the program.

→ All statements in the 'declaration' & 'executable' part end with 'semicolon'.

⑥ Subprogram section :-

→ The subprogram section contains all the 'user-defined functions' that are called in the main() function.

→ 'User-defined functions' are generally placed immediately after the 'main() function', although they may appear in any order.

Example :-

```
# include <stdio.h>
int main( )
{
    Printf("Welcome to c Programming\n");
    return 0;
}
```

Output:-

Welcome to c programming.

* C-Tokens:-

→ Building blocks for writing/creating a C-program are called 'C-Tokens'.

→ Each instruction/statement can write by Token.

Ex:- int i=10 ;

classification:-

① Identifiers.
② Variables.
③ Keywords.
④ Data Types.
⑤ constants
⑥ Operators.

① Identifiers :-

→ 'Name' given to 'programming element' is called 'Identifer'.

Name
↓
user defined words
↓
combination of upper/lower/digits/_underscore

Programming element
↓
variables, arrays, strings, pointers, etc—

Ex:- float a=20.2;

↓

Identifier

_cse ✓

R_cse ✓

9cse ✗

R@cse ✗

→ shouldn't allow special character.

Rules:-

① starts with 'alphabets only' [shouldn't starts with digits]

② 'spaces' not allowed in a name.

③ shouldnot use special symbols except 'underscore'

④ shouldnot match with 'keywords'

⑤ Length of Identifier ≤ '31 characters'

⑥ names must be 'meaningful' & 'essay'

⑦ 'case sensitive', A ≠ a, X₁ ≠ x₁

⑧ Identifier should be 'unique'

A-65

a-97

0-48

② variables :-

→ Giving name to 'memory location'.

→ It is a memory location used to store some data/values

→ values may change thus called 'variables'.

Syntax:-

① int a = 10;

↓ datatype → variable

② int a, b, c;

→ variable Declaration — specific to particular program

③ Keywords:-

→ These are also called as 'Reserved words' (or) 'pre defined words'.

→ All keywords have 'fixed meanings' and these meanings cannot be changed.

→ There are '32 keywords' in C Programming.

→ Keywords serve as basic building blocks for a program statement

→ All keywords must be written in 'lowercase' only.

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	size of	volatile
do	if	static	while.

④ Data types:-

→ The type of data which we need to store in memory known as 'Data types'.

→ There are '3 types' of Datatypes.

① Primitive Data types:- 'char', 'int', 'float', 'double', 'void'
→ It allows to store only single value.

② Derived Data types:- 'string', 'arrays', 'pointers' etc.
→ It allows to store multiple values of same type.

③ User Data types:- 'structure', 'union', 'typedef', 'enum' etc
→ combination of both Primitive & derived data types.

Note :-

→ 'Primitive data types' are Basic data types

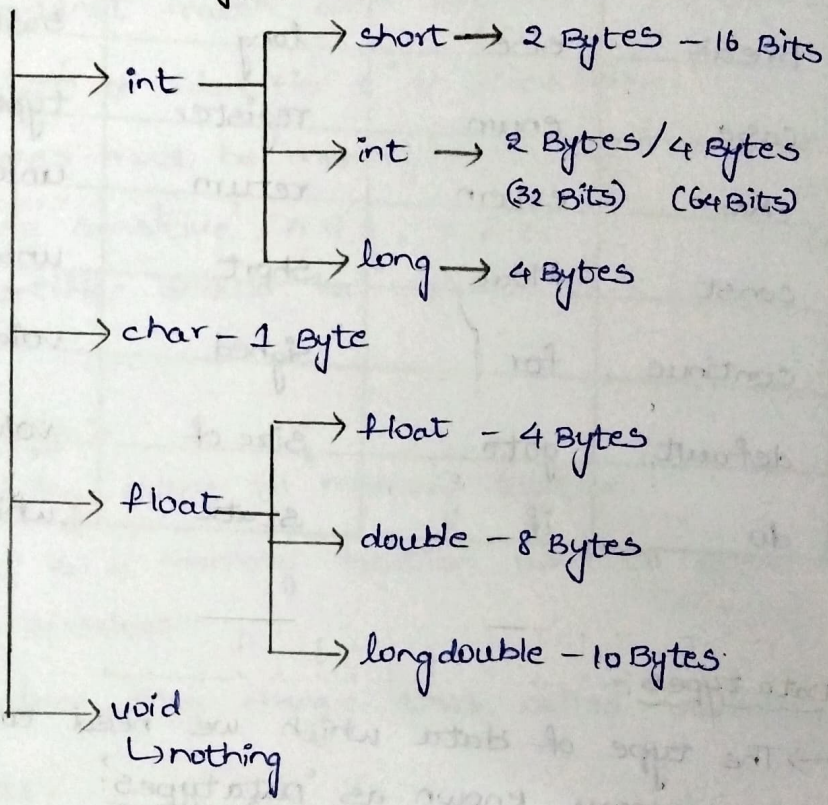
→ 'Derived data types' are 'collection of Basic data types'. They are derived by Primitive D.types

Ex: 'string' is collection of 'characters'

→ 'Primitive' & 'Derived data types' are defined by the program developers.

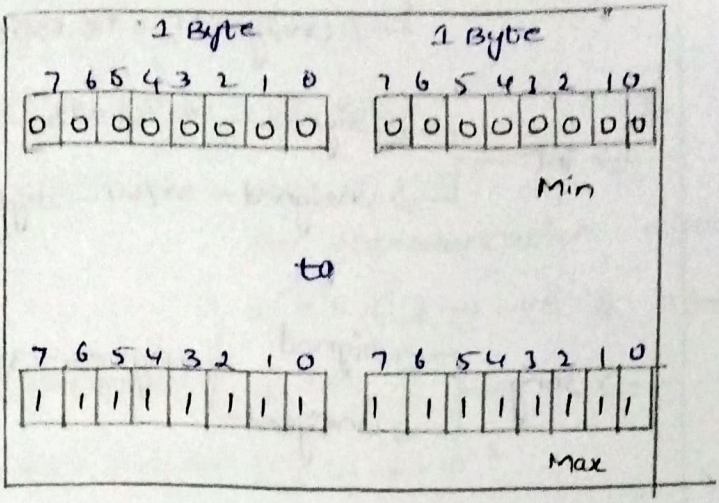
→ 'user data types' are defined by user.

① Primitive Data types :-



10 → 1010 2 | 10 2 | 5 – 0 2 | 2 – 1 1 – 0 1010 (4 Bits)	2 | 36 2 | 18 – 0 2 | 9 – 0 2 | 4 – 1 2 | 2 – 0 1 – 0 36 → 100100 (6 Bits)	2 | 512 2 | 256 – 0 2 | 128 – 0 2 | 64 – 0 2 | 32 – 0 2 | 16 – 0 2 | 8 – 0 2 | 4 – 0 2 | 2 – 0 1 – 0 1000000000
1 0 1 0 × × × × $2^3+2^2+2^1+2^0$ 8+0+2+0 = 10	1 0 0 1 0 0 × × × × × × $2^5+2^4+2^3+2^2+2^1+2^0$ 32+0+0+4+0+0 = 36	1000000000 2^9+0 → $(512)_{10}$

(1)* int (short) :-



00000000 00000000 → $(0)_{10}$

11111111 11111111 → $(65535)_{10}$

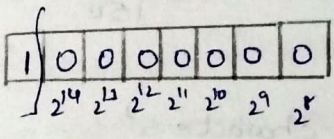
2^{15} ... 2^0

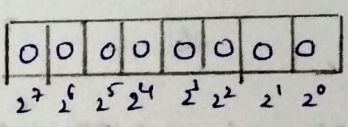
65536 → 1 00000000 00000000 → result is '0'

8 bits 8 bits

Data → signed → -ve, 0, +ve values

Data → unsigned → 0, +ve values



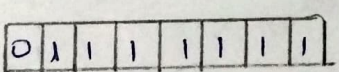


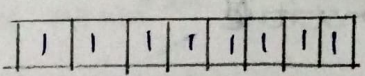
1 → -ve

0 → +ve

2^0 to 2^{14} & $(-1)=2^{15}$

⇒ -1 to -32768 (minimum).



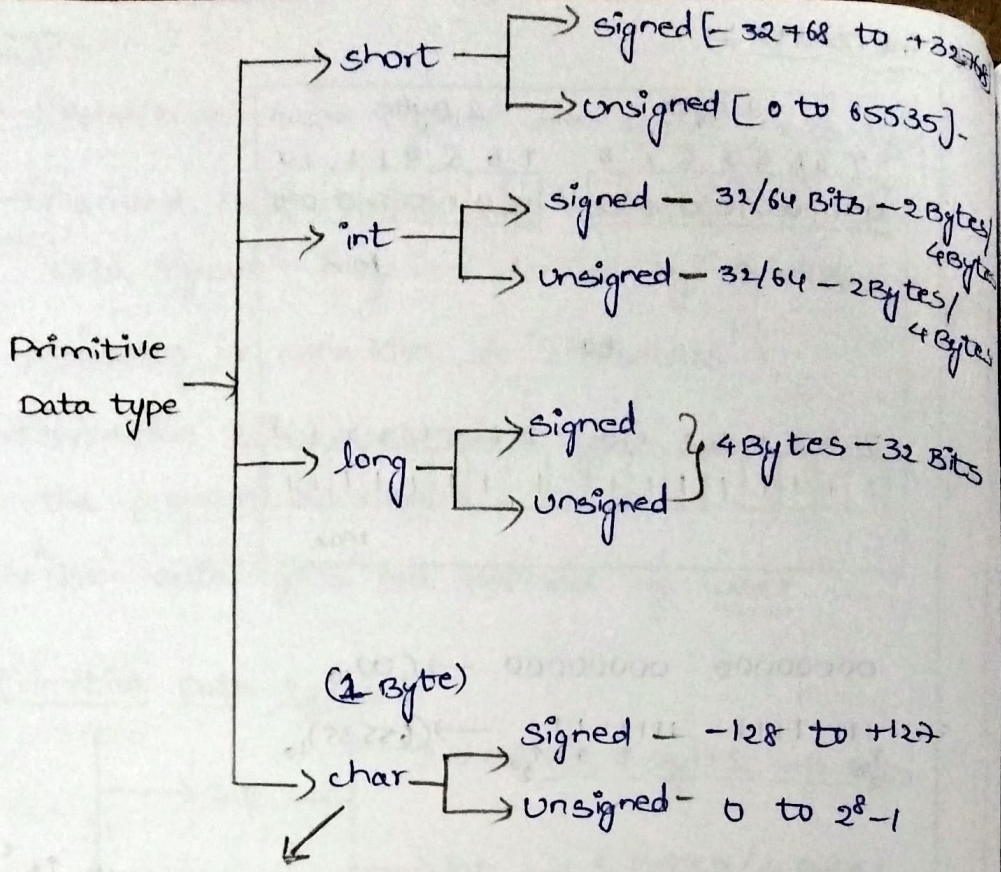


0 to 32768 (Maximum)

For unsigned :- 0 to $2^{n}-1$ ⇒ 0 to $2^{16}-1$ ⇒ 0 to 65535

For signed :- -2^{n-1} to $+2^{n-1}-1$

⇒ -2^{15} to $+2^{15}-1$ ⇒ -32768 to 32768



(2)* char :-

* Input: characters ⟶ Machine language
Numerical 0 & 1's

A-65	a-97	0-48	Special char
B-66	b-98	1-49	#-35
⋮	⋮	⋮	Space-32
Z-90	Z-122	9-57	⋮
26	26	10	150

Total = 26 + 26 + 10 + 150 = 212 characters.

ASCII: American standard code for Information Institute.
↓
developed by unicode system.

→ To store these 212 characters, 8 bits are required. i.e, 1 Byte.

* float :-

(3)

(i) float :-

→ 4 Bytes (32 Bits)

→ Montisa :- +ve or -ve

for representation allocated.

1 bit (1 → -ve & 0 → +ve).

→ Exponent :- 8 bit

→ Ex:- 1.01 ⇒ $\frac{101}{100}$ ⇒ 101×10^{-2}

Montisa → '1 bit'

Exponent → '8 bits'

Value → '23 bits'

Precision → '7 decimal digits' (2.1111111)

→ Range :-

1.7E-38 to +3.4E+38

1.7×10^{-38} to $+3.4 \times 10^{+38}$

(ii) Double :-

→ 8 bytes - 64 bits

→ Montisa → '1 bit'

→ Exponent → '11 bits'

→ Value → '52 bits'

→ Precision → '15 decimal digits'

→ Range :- 2.3E-308 to 1.7E+308

2.3×10^{-308} to $1.7 \times 10^{+308}$.

(iii) long double :-

→ 12 Bytes

→ Montisa → '1 bit'

→ Exponent → '15 bits'

→ Value → '80 bits'

→ Precision → '21 decimal digits'.

* Data types:-

Data type	size (in bytes)	Specifier	Range
① char	1	%c	-128 to 127
② signed char	1	%c	-128 to 127
③ unsigned char	1	%c	0 to 255
④ Int	2	%d	-32768 to 32767
⑤ signed int	2	%d	-32768 to 32767
⑥ unsigned int	2	%u	0 to 65535
⑦ short int	2	0	-32768 to 32767
⑧ signed short int	2	0	-32768 to 32767
⑨ unsigned short int	2	0	0 to 65535
⑩ long int	4	%ld	-2147483648 to 2147483647
⑪ signed long int	4	%ld	-2147483648 to 2147483647
⑫ unsigned long int	4	%lu	0 to 4294967295
⑬ float	4	%f	3.4E-38 to 3.4E+38
⑭ double	8	%lf	1.7E-308 to 1.7E+308
⑮ long double	10	%Lf	3.4E-4932 to 1.1E+4932.

*⑯ Void:-

→ This type holds no value.

→ It is used to

(i) to specify the return type of a function [When the function returns no value].

(ii) to specify the parameters of the function [When the function accepts no arguments from caller]

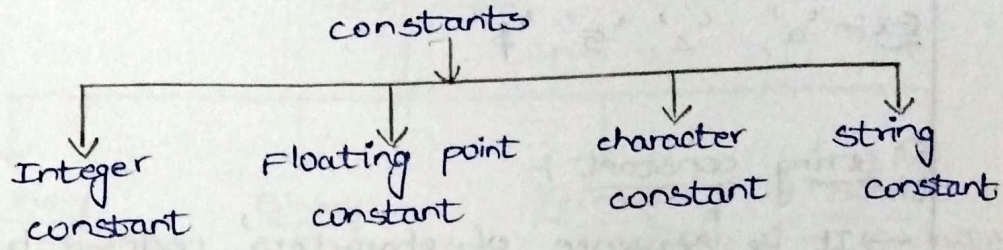
(iii) to create generic pointers.

5 Constants :- (or) Literals :-

→ It is a value that can be stored in memory & cannot be change during execution of program.

→ constants are used to define 'fixed values'.

Ex :- Pi



(i) Integer constant :-

→ must have atleast 'one digit'.

→ shouldnot have a 'decimal point'.

→ can be '+ve' or '-ve'.

Ex :- 1, 9, 234, -999.

(ii) Floating point constant :-

→ It is a 'decimal number'.

→ It contains either 'decimal point' or 'exponent'.

In Fractional form :-

→ should be atleast '1 digit'

→ could be '+ve' or '-ve'

→ 'decimal point' is must.

→ No 'commas' or 'blanks'.

In Exponential form :-

→ consists two parts → Before e - 'Mantissa'
→ after e - 'exponent'

→ Mantissa & exponential seperated by 'E'.

→ Mantissa can have a '+ve' or '-ve' sign. [default +ve]

Ex :-

2E-10., 0.5e2, 1.2E+3, -5.6E-2.

(iii) character constant :-

→ These are 'single character' enclosed in 'single quotes'.

→ It can be 'single alphabet', 'single digit', 'single special symbol' enclosed in 'single quotes'.

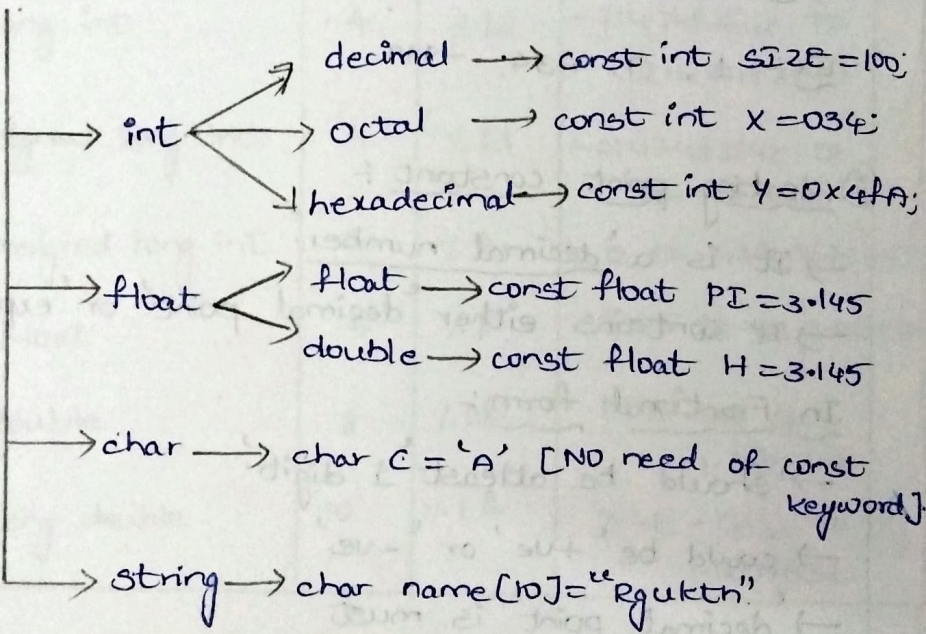
Ex :- 'a', 'z', '5', '$'

(iv) string constant :-

→ It is 'sequence of characters' enclosed by 'double quotes'.

Ex :- "hello", "cse", "a" ≠ 'a'.

* Constants in Brief :-



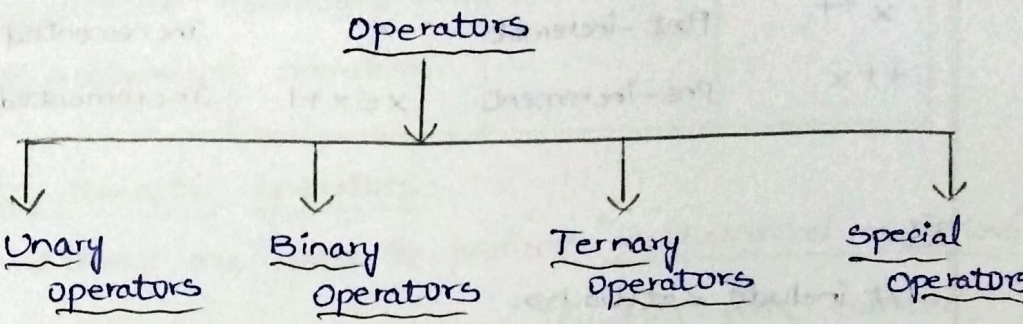
→ As a good programmar, we need to use constants in 'capital letters'.

* Literals :- The fixed value that is going to be initialized (or) assigned directly in to variable after declaration is called 'Literal'.

Ex :- int x=10 ; → Literal [The value may change]
const int x=10; → constant [Value may not change]

⑥ Operators :-

→ An 'operator' is defined as a 'symbol' that 'operates on 'operands' and does operations like 'mathematical', 'relational', 'logical' etc.



(i) Unary operators :-

→ An operator which operates on single operand is called 'Unary operator'.

→ C supports '3 unary operators'.

ⓐ 'Unary minus'

ⓑ 'Increment operator'

ⓒ 'Decrement Operator'.

ⓐ Unary minus :-

→ It returns the operand multiplied by '-1'.

→ It changes the 'sign'

Ex :- int a, b = 10

a = -(b)

Result :- 'a = -10'

ⓑ Increment operator :-

→ It increases operand's value by 1.

→ There are 2 ways to use increment operator.

Post increment :- If you put the operator after the operand (postfix), it returns the original value of operand [Before increment].

Pre increment :- If you put operator infront of operand (prefix), it returns new value of operand [incremented].

Operator	Name	Value returned	Effect on variable
x ++	Post-increment	x	Incremented
++x	Pre-increment	x = x+1	Incremented

Ex :-

```
# include <stdio.h>
int main( )
{
    int x=5;
    Printf("x=%d\n", x++);
    Printf("x=%d\n", ++x);
    return 0;
}
```

Output
x=5
x=7

© Decrement operator :-

→ It decreases value of operand by '1'

→ It also has 'Post decrement' & 'pre decrement'

Operator	Name	Value returned	Effect on variable
x --	Post--decrement	x	Decremented
--x	Pre--decrement	x = x-1	Decremented

Ex :-

```
# include <stdio.h>
int main( )
{
    int x=5;
    Printf("x=%d\n", x--);
    Printf("x=%d\n", --x);
}  return 0;
```

Output
x=7
x=5

(ii) Binary operators :- [operate on two operands at a time]

(a) Arithmetic operators

(b) Relational operators

(c) Logical operators

(d) Bitwise operators

(e) Assignment operators.

(a) Arithmetic operators :-

→ These are used to perform 'mathematical operations'

→ Applied to any 'integers', 'floating-point number', 'characters'.

→ 5 arithmetic operators :- '+', '-', '*', '/', '%'

→ '%' can be applied to integer operands only and cannot be used on 'float' or 'double values'.

Ex :- int a=9, b=3, result; Here a & b are operands

Operation	operator	syntax	statement	Result.
① Addition	+	a+b	result=a+b	12
② substraction	-	a-b	result=a-b	6
③ Multiply	*	a*b	result=a*b	27
④ Divide	/	a/b	result = a/b	3
⑤ Modulo	%	a%b	result=a%b	0

Ex :-

```
#include <stdio.h>
int main()
{
    int a=9, b=3;
    Printf("%d + %d = %d\n", a,b, a+b);
    Printf("%d - %d = %d\n", a,b, a-b);
    Printf("%d * %d = %d\n", a,b, a*b);
    Printf("%d / %d = %d\n", a,b, a/b);
    Printf("%d %% %d = %d\n", a,b, a%b);
    return 0;
}
```

output :-

```
9+3=12
9-3=6
9*3=27
9/3=3
9%3=0
```

(b) Relational operators :- [comparison operator]

→ It 'compares' two operands.

→ Operands can be 'variables', 'constants' or 'expressio

→ Always return either 'True' or 'False'

Operator	Meaning	Example
<	Less than	3<5 gives 1
>	Greater than	7<9 gives 0
<=	Less than or equal to	100<=100 gives 1
>=	Greater than or equal to	50>=100 gives 0
== (equality operator)	Equal to	10==9 gives 0
!= (equality operator)	Not equal to	10!=9 gives 1

Ex:-

```
# include <stdio.h>
int main( )
{
    int a=9, b=3;
    Printf("%d >%d =%d\n", a,b, a>b);
    Printf("%d < %d =%d\n", a,b, a<b);
    Printf("%d <= %d =%d\n", a,b a<=b);
    Printf("%d >= %d = %d\n", a,b, a>=b);
    Printf("%d == %d = %d\n", a,b, a==b);
    Printf("%d != %d = %d\n", a,b, a!=b);
    return 0;
}
```

output :-

```
9 > 3 = 1
9 < 3 = 0
9 <= 3 = 0
9 >= 3 = 1
9 == 3 = 0
9 != 3 = 1
```

© Logical operators :-

→ used to combine 2 or more relational expressions

→ 3 logical operators

AND (&&) → } 'Binary'
OR (||) → }
NOT (!) → 'Unary'

→ returns either '0' (False) or '1' (True).

AND :-

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

NOT

A	!A
0	1
1	0

Ex :-

```
#include <stdio.h>
int main( )
{
  int z1, z2, z3;
  z1 = 7>5 && 10>15;
  z2 = 7>5 || 10>15;
  z3 = ! (7>5);
  Printf ("z1 = %d\n z2 = %d\n z3 = %d\n", z1, z2, z3);
  return 0;
}
```

output :-

z1 = 0

z2 = 1

z3 = 0

④ Bitwise Operators :-

→ Bitwise operators operate on 'Bits'

→ These operators include :

bitwise AND (&)

bitwise OR (|)

bitwise X-OR (^)

bitwise NOT (~)

bitwise shifts < shift left (<<) / shift right (>>)

Bitwise AND (&) :-

Ex :- int a=6, b=9, c;

c=a&b;

Printf("%d", c); //prints '0'.

0110 [6]
0001 [9]
0000 [0]

Bitwise OR (|) :-

Ex :- int a=4, b=2, c;

c=a|b;

Printf("%d", c); // prints '6'.

100 [4]
010 [2]
110 [6]

Bitwise XOR (^) :-

Ex :- int a=4, b=2, c; //prints '6'

c=a^b;

print("%d", c);

100 [4]
010 [2]
110 [6]

XOR

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise NOT (~) :-

Ex :- int a=4; c;

c=~a;

Printf("%d", c);

~100 [4]
101 [5]

Shift operators:-

Operand	operator	num

Left shift :-

Ex:- 2^3 2^2 2^1 2^0
0 1 0 0

x<<1 = 1 0 0 0 // result is 8 (1000)

→ 'Left-shift' is equals to 'multiplication by 2'

Right shift :-

Ex:- 8 4 2 1
2^3 2^2 2^1 2^0
0 1 0 0

0 0 1 0 // x>>1 is 2(10)

→ 'Right-shift' is equals to 'division by 2'.

Example for Bitwise operators :-

```
#include <stdio.h>
int main()
{
  int a=4, b=2;
  Printf("%d & %d = %d \n", a,b,a&b);
  Printf("%d | %d = %d \n", a,b,a|b);
  Printf("%d ^ %d = %d \n", a,b,a^b);
  Printf("~%d = %d \n", a,~a);
  Printf("%d <<1 = %d \n", a,a<<1);
  Printf("%d >>1 = %d \n", a,a>>1);
  return 0;
}
```

output :-

```
4&2 =0
4|2 =6
4^2 =6
~4 = -5
4<<1 = 8
4>>1 = 2.
```

⑤ Assignment operators:-

left side (variable) = Right side (value) or location

Ex: ① int x=2, y=3, sum;

sum = x+y; 5

② int a=b=c=10

(a=(b=(c=10)))

Operator	Description	Example
+=	Add AND	C+=A (or) C=C+A
-=	Substract AND	C-=A (or) C=C-A
*=	Multiply AND	C*=A (or) C=C*A
/=	Divide AND	C/=A (or) C=C/A
%=	Modulus AND	C%=A (or) C=C%A
<<=	Left shift AND	C<<=2 (or) C=C<<2
>>=	Right shift AND	C>>=2 (or) C=C>>2
&=	Bitwise AND	C&=2 (or) C=C&2
^=	Bitwise exclusive OR	C^=2 (or) C=C^2
|=	Bitwise inclusive OR	C|=2 (or) C=C|2.

Ex:-

```
#include <stdio.h>
int main()
{
    int a=21, c;
    c=a;
    Printf("operator is = and c=%d \n", c)
    Printf("operator is += and c=%d \n",c)
    Printf("operator is -= and c=%d \n",c)
    Printf("operator is *= and c=%d \n",c)
    Printf("operator is /= and c=%d \n",c)
}
```

output:-

```
21
42
21
441
21
```

(iii) Ternary or conditional operators :-

→ It operators on three operands.

variable = expression 1 ? exp 2 : exp 3

Ex :-

a, b

① Max = (a>b) ? a : b

True False

② even / odd

n%2 ? printf("%d is odd", n) : printf("%d is even", n);

True False

③ a, b, c (Nested)

Max = (a>b) ? (a>c) ? a : c : (b>c) ? b : c :

T F T F

True False

(iv) Special operators :-

'*' → dereference / Indirect operator

'&' → Address of memory.

'.' → period operator [For member accessing operator]

'→' → pointer-member accessing operator

'sizeof()' → No. of bytes allocated for variable/datatype

'comma' → Multiple expressions

* Expressions :-

① simple :-

→ contains one operator & 2 operands

Ex :- x+y; 3+5; a*b; x-y etc.

② complex :-

→ contains 2 or more operators & operands

Ex :- x+y-2, a+b-c*d, 2+5-3*4, x=6-4+5*2 etc.

* Type conversion :-

→ To convert variable from one 'data type' to 'another data type'.

Implicit :-

→ performed by the 'compiler' without 'programmer intervention'.

→ All 'Lower data types' converted to next 'higher data types'.

Rules :-

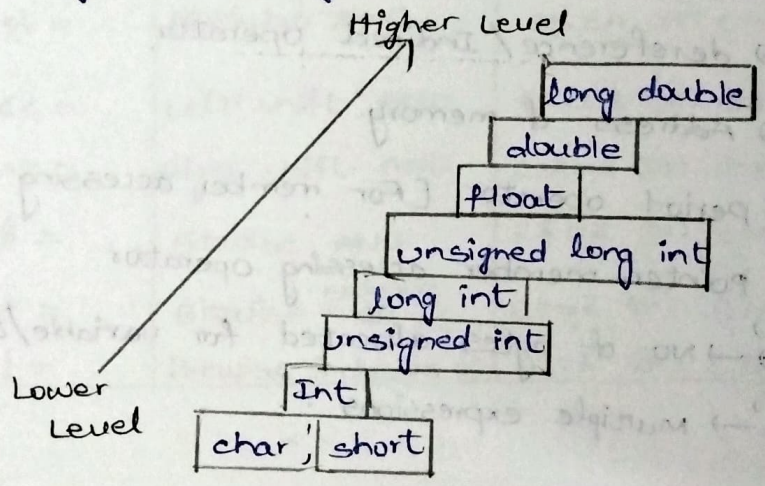
→ 'float' converted to 'double'

→ 'char' or 'short' converted to 'int'.

→ anyone is 'double' then other is also 'double'.

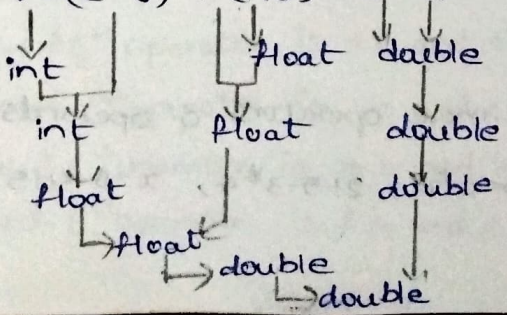
→ anyone is 'long' then other converted to 'long'.

→ anyone is 'unsigned' other is also 'unsigned'.



Ex :- char c;
int j;
float f;
double d, r;

r = (c * j) + (f / j) - (f + d).



Explicit :- [Type casting]

→ performed by programmar for his requirement

→ syntax :- (data_type) expression;

Rules :-

→ All 'integers' converted to 'float'

→ 'float' types → 'double'

→ 'character' → 'integer'

Ex:

① a = int(9.5); 9

② a = int(12.3) / (int) 4.2; 12/4 = 3

③ a = (double) total / n; float

④ a = (int) (a+b); int

* Precedence and Associativity :-

S.NO	operators	Associativity
①	( ), [ ], →, ., ++, --	Left to Right
②	++, --, !, ~, sizeof( ), + (unary), - (unary), & (address), * (indirection)	Right to Left
③	*, /, %	Left to Right
④	+, -	Left to Right
⑤	<< >>	Left to Right
⑥	<<= >>=	Left to Right
⑦	==, !=	Left to Right
⑧	&	Left to Right
⑨	^	Left to Right
⑩	|	Left to Right
⑪	&&	L to R
⑫	||	L to R
⑬	? :	R to L
⑭	=, +=, -=, *=, /=, %=, >>=, <<= &=, ^=, |=	R to L
⑮	,	L to R.