C Programming

Important Points to Remember -
-Portable - can run our code on any machine.
-Compiler Translates C code into Binary code.
-Procedural - Divide big task into smaller Task/Functions
-Pre-processor Directive -
replaces code after # symbol with actual code file before compilation occurs
Eg. - #include <stdio.h>
Source code (1)--> Expanded Code -->(2) Machine code
1 - Preprocessor 2 - Compiler
-Header File - files with .h extension
Contains Declarations/Prototypes of some standard functions

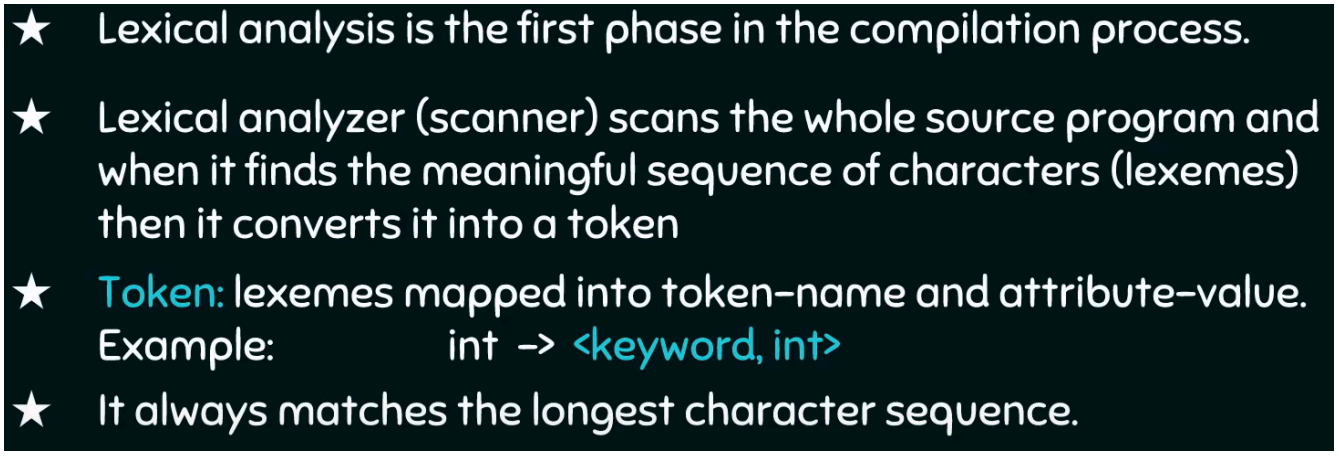
- C Library contains actual code/Defination of Functions , whose declarartions are in Header File.
- Linker links(or maps) source code with actual defination of input functions , stored in std. C library. It doesn't copy Paste entire code /Definition
- main is Starting Point of a C program
  We can give run type arguement in C called argc and argv.
- Sizeof()- its an Unary operator and not a functn
  Ut returns size of input datatype.
  Eg. - sizeof(int) --> 4 bytes
- short unsigned <----> unsigned short
- Any value starting with 0 is an Octal value.
  Eg. - int var = 052 --> Here var is = 42 in decimal
  $$\begin{matrix} 8^1 & 8^0 \\ 5 & 2 \end{matrix} \qquad 5*8+2*1=40+2=42$$
- Any value starting with 0x is a Hex value
  Eg. - int var = 0x43ff
  Note - %x will give - 43ff
  but %X will give 43FF
  - Lvalue means an entity(usually variable) which has an dedicated memory location and can store some data
    They appear on LHS of = operator.
    Eg. - int x = , *ptr = ,
    whereas Rvalue is an entity which is temporary and don't have an identifiable address and can't

store data

They appear on RHS of = operator.

Eg. - = 10, = 5+3

- Lexical Analysis -



Tokens Generation -

- About SizeOf() operator -

The sizeof operator yields the size (in bytes) of its operand, which may be an expression or a parenthesized name of a type. The size is determined from the type of the operand. If the type of the operand is a variable length array type, then the operand is evaluated; otherwise, the operand is not evaluated and the result is an integer constant.

So, here `int var = sizeof(i++);`

i++ won't be evaluated.

List of Header Files -

<assert.h>	<math.h>	<stdlib.h>
<complex.h>	<setjmp.h>	<stdnoreturn.h>
<ctype.h>	<signal.h>	<string.h>
<errno.h>	<stdalign.h>	<tgmath.h>
<fenv.h>	<stdarg.h>	<threads.h>
<float.h>	<stdatomic.h>	<time.h>
<inttypes.h>	<stdbool.h>	<uchar.h>
<iso646.h>	<stddef.h>	<wchar.h>
<limits.h>	<stdint.h>	<wctype.h>
<locale.h>	<stdio.h>	

- stdio.h - contains std I/O functn like printf, scanf
- limits.h - contains CONSTANTS to find max, min values of different data types

  Eg- INT_MIN, SHRT_MAX - for signed

UINT_MAX , USHRT_MAX - for unsigned(UINT_MIN is not defined b/c min value is 0 only for unsigned int)

- stdlib.h - contains malloc(), realloc() and memory management, program termination, and conversion functions(atoi ...etc)
-

How to run code -

To compile - gcc <file.c> -o

and then .<file>

Types of Error Messages -

- If we remove # from header files

```
prac.c:1:9: error: expected '=', ',', ';', 'asm' or '__attribute__' before '<' token
 include <stdio.h>
         ^
```

- If we remove header file stdio.h

```
PS C:\Users\AMIT HIWAL\Desktop> gcc prac.c -o prac
prac.c: In function 'main':
prac.c:5:5: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
    printf("Neso Academy!!");
    ^~~~~~
prac.c:5:5: warning: incompatible implicit declaration of built-in function 'printf'
prac.c:5:5: note: include '<stdio.h>' or provide a declaration of 'printf'
```

- If removed Semicolon after any statement

```
prac.c: In function 'main':
prac.c:5:5: error: expected ';' before 'return'
     return 0;
     ^~~~~~
```

- If removed return Type int from int main

```
prac.c:2:2: warning: return type defaults to 'int' [-Wimplicit-int]
  main ()
  ^~~~
```

- If returned any other interger or character instead of 0

  Nothing Happens, Program execute successfully

- If removed semicolon after return 0

```
prac.c: In function 'main':
prac.c:6:1: error: expected ';' before '}' token
 }
 ^
```

Variables -

- Name given to a memory location to store data in it.
- Declaration - announcing properties to compiler

  -Definition - allocating memory for variable

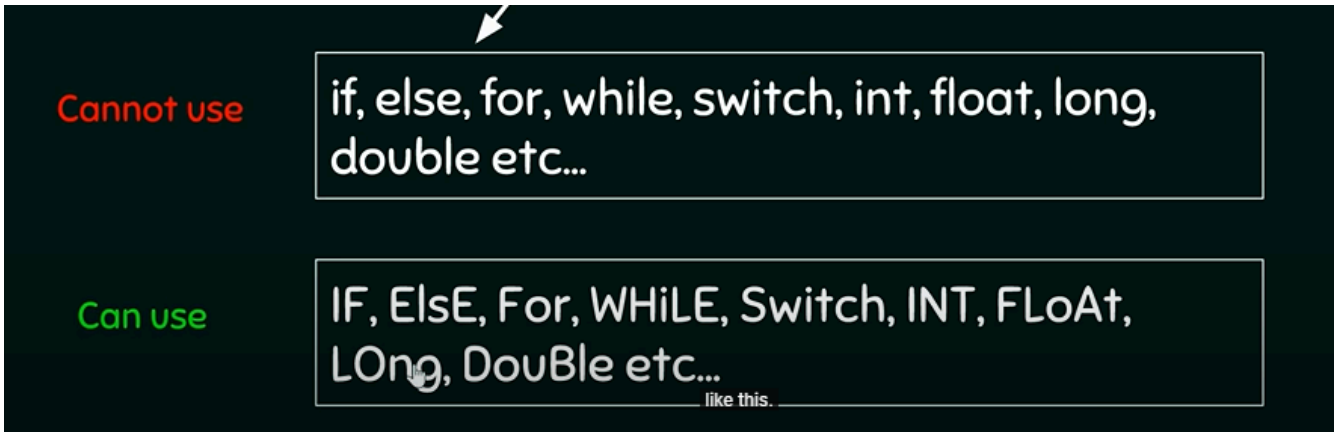
- Initialization - assiging value to a variable
  `Syntax - <datatype> <name> = <value>;`
  Eg. - `int var = 3;`
- To change its value , `var =4;`
  Inside main, we can't define 2 variables with same name.
- Datatype indicates how much size is req for that variable in memory.
- var can be initialized with another variable too.

```
Eg. - int var1 =3;
          int var2 = var1;
```

- Define multi-variables in single line -

```
Eg. -  int var1, var2, var3;
          var1=var2=var3=7;
```

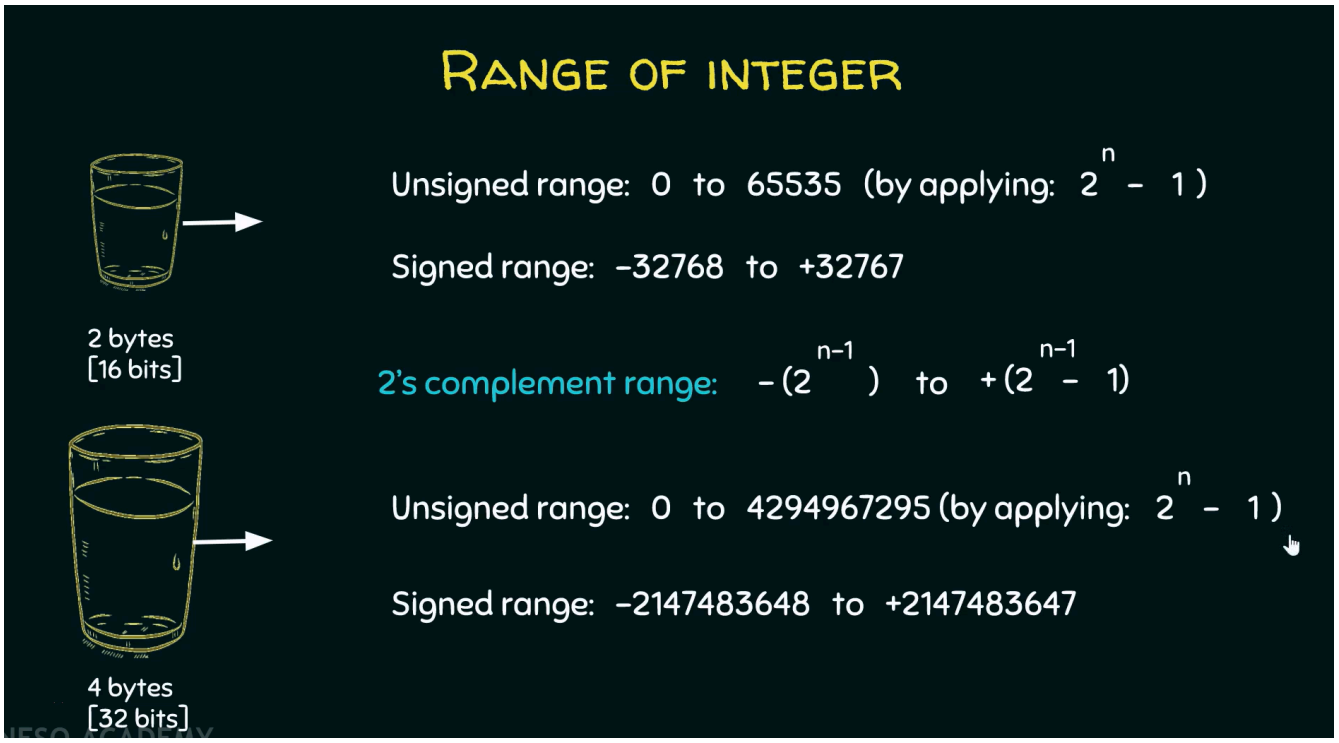
- Naming Convention -
  can't start with number, underscore(_)
  Names are case sensitiveand we generally use smallcase for variables.
  Don't use special characters except underscore(but not at beginning)
  Don't use whitespaces in betweeen variable names.
  Don't use Keywords - use only with atleast 1 char Uppercase

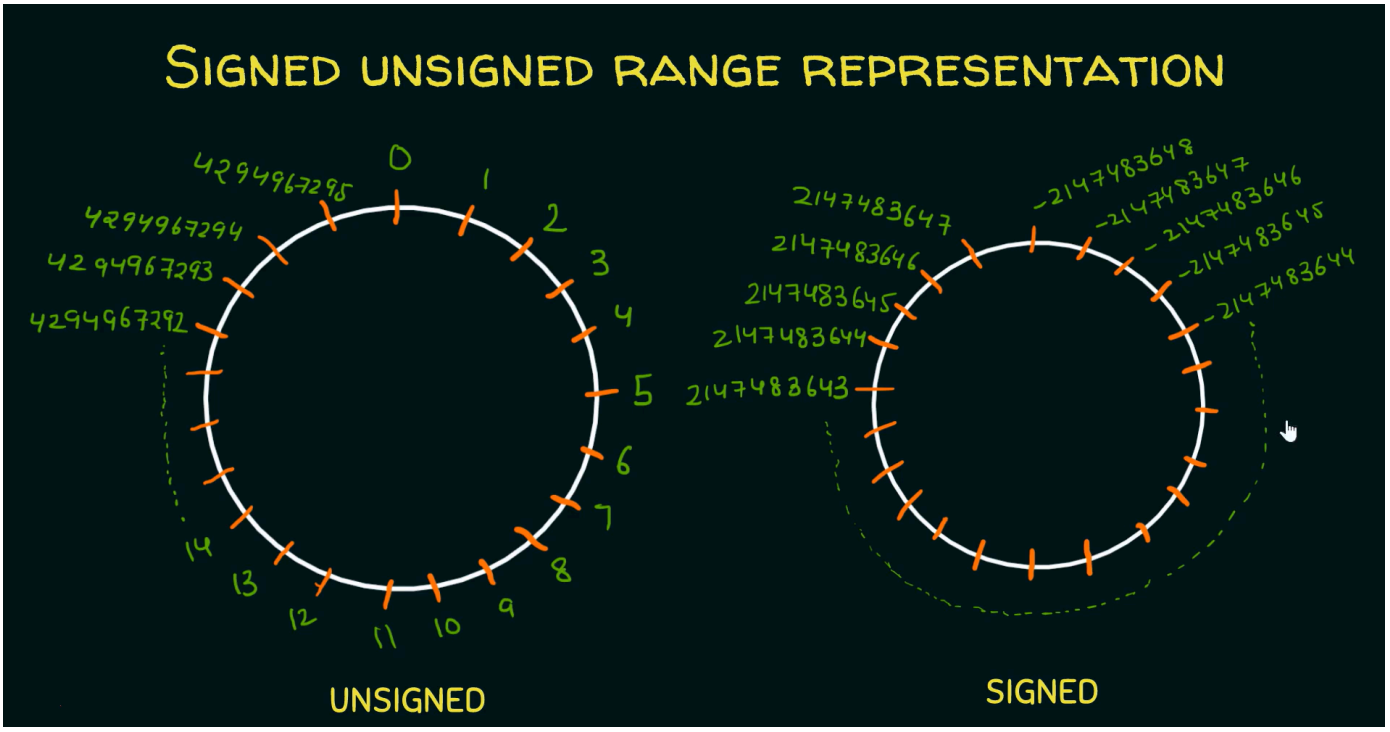


DataTypes Supported in C-

- By Default, compiler will assume int if not defined.

1. Integer -
   Range - follows mod2^n function





Modifiers on int -

short - 2 bytes

int - 2/4 bytes

long - 4/8 bytes

long long - 8bytes

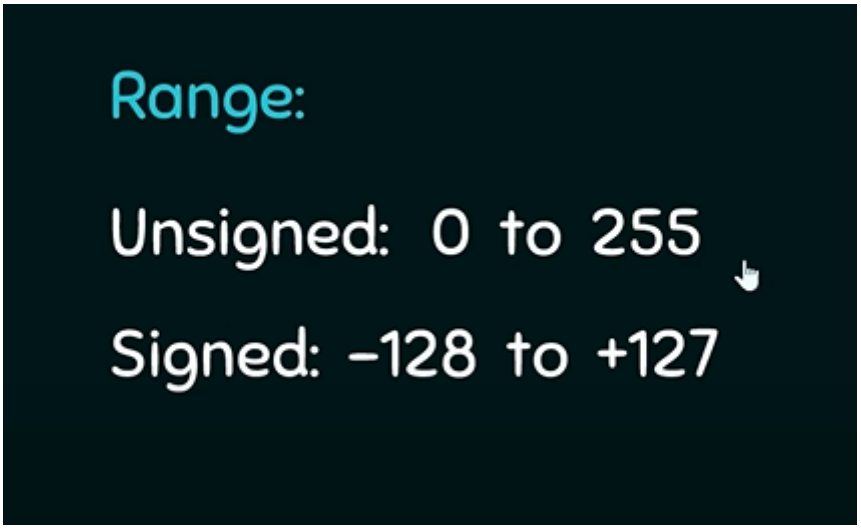
signed(DEFAULT) - Both positive and negative value

unsigned - only Positive value

2. char - 1 byte = 8bits

Eg . - char var = 'A' **or** 65

Range -



ASCII Table - 7bits/char

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Extended ASCII - 8bits/char

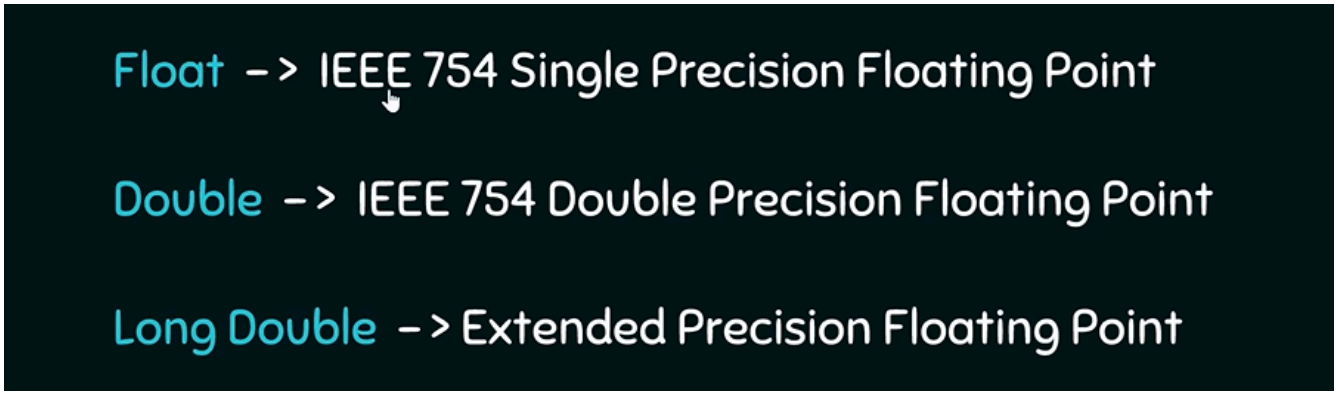
ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

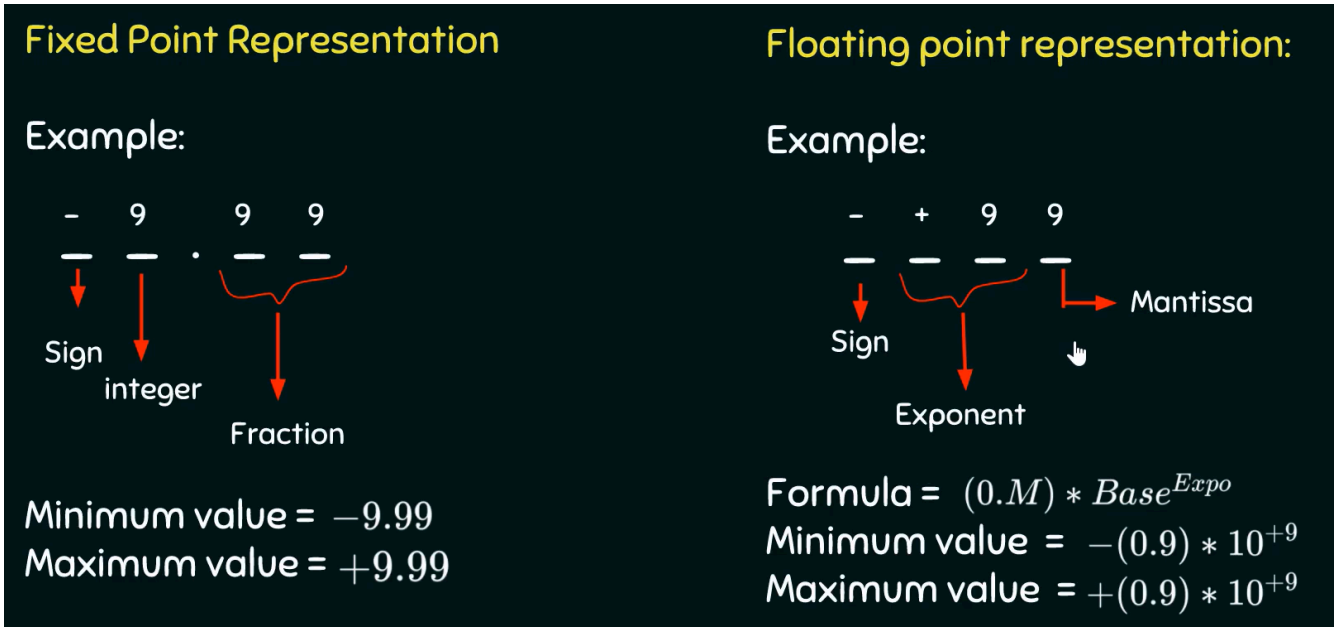
Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

3. float/double - to asssign Fractional values

- Size -
  float - 4 bytes --> 7 digits(including before decimal)
  double - 8 bytes --> 16 digits
  long double - 12 bytes --> 19 digits
  Represenations -



Fixed V/S Floating Point Representations -



PlaceHolders -

%hd - signed short
%hu - unsigned short
%d - signed int
%lld - long long signed int
%o - octal value
%x, %X- hex value(lowercase), hex value(uppercase)
%u - unsigned int
%ld, %lu - (signed long int, unsigned long int)
%llu - unsigned long long int
%f - float/double
%.7f - float with 7 digits precision (after decimal)
%lf, %.8lf - double , double with 8 digits precision(after .)
%Lf, %.21Lf - long double , long double(w/ 21 digits precision afer decimal)
%c - char
%s - string value
%10s - string with 10 char(If less char are i/p, then add whitespaces on MSB side)
%zu - size_t
%p - pointer address

Comments -

// - for single line comments
/* ---------
---------- --> For Multiline Comments
----------*/

Output -

printf("Hello %s", string)

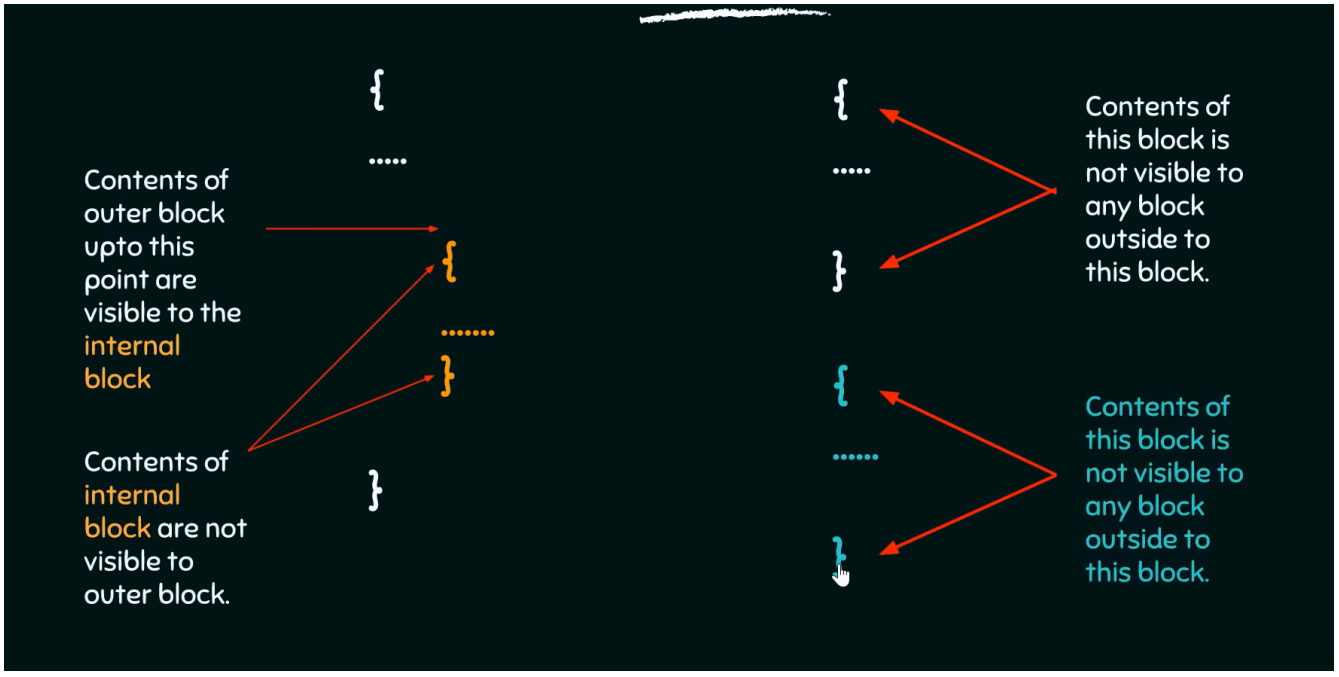
- printf prints the characters and also returns number of characters it successfully prints
- If want to print `%`, then use %% b/c one % we use for placeholders.

Input -
scanf("%d", &var)

- takes strings, numeric values , special characters from user and store in var.
- & - address of operator, returns address of any var.
- `scanf("%[^\n]s", str);` - reads until \n is encountered.

Scope of Variables -
Local Variables -

- variables defined inside a block of code/functn are automatically destroyed after the functn/block execution ends and hence can only be accessed within that block/functn.
- Such variables are called as local variables



- If we have same var in both internal and outer block(above internal block), then preference will be given to immediately available variable i.e. internal var
- If uninitialized, it will take a garbage value.
  Global Variables -
- variables defined in a program(test.c), usually below header files, outside of main or any function are called Global variables.
- Global variabes are accessible to anything inside that test.c file and to any other file in the project.
- If uninitialized, by default it will be given value 0.

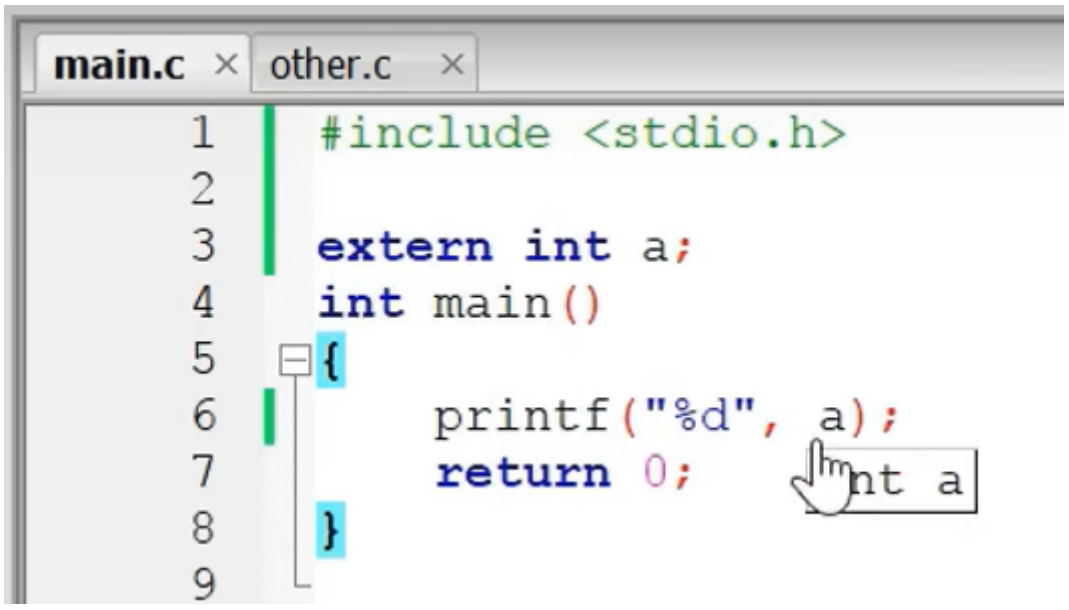
Modifiers in C -
By default, compiler assumes as auto type variable

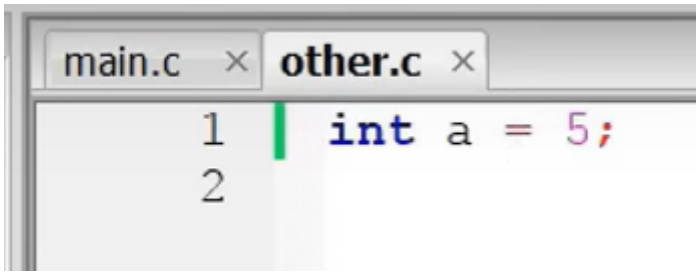
1. auto - automatically destroyed after its scope

- If uninitialized, it will take a garbage value.
- Multiple declaration is not allowed.

2. extern - declare but don't allocate space to var
   `extern int a;`

- used when we want to use var defined in some other file,(must be global), or even outside the scope of current block in current file also.





- Multiple declararion allowed ,within that file, for extern type var.
- If defined also, then memory is allocated
  `extern int a = 9;`
- If value not found outside, Linker will throw error. Compiler will asssume OK.

3. register - hints compiler to allocate space in register for this var. compiler may or may not do so.

- Usually, frequently used var, compiler allocates space into register automatically.

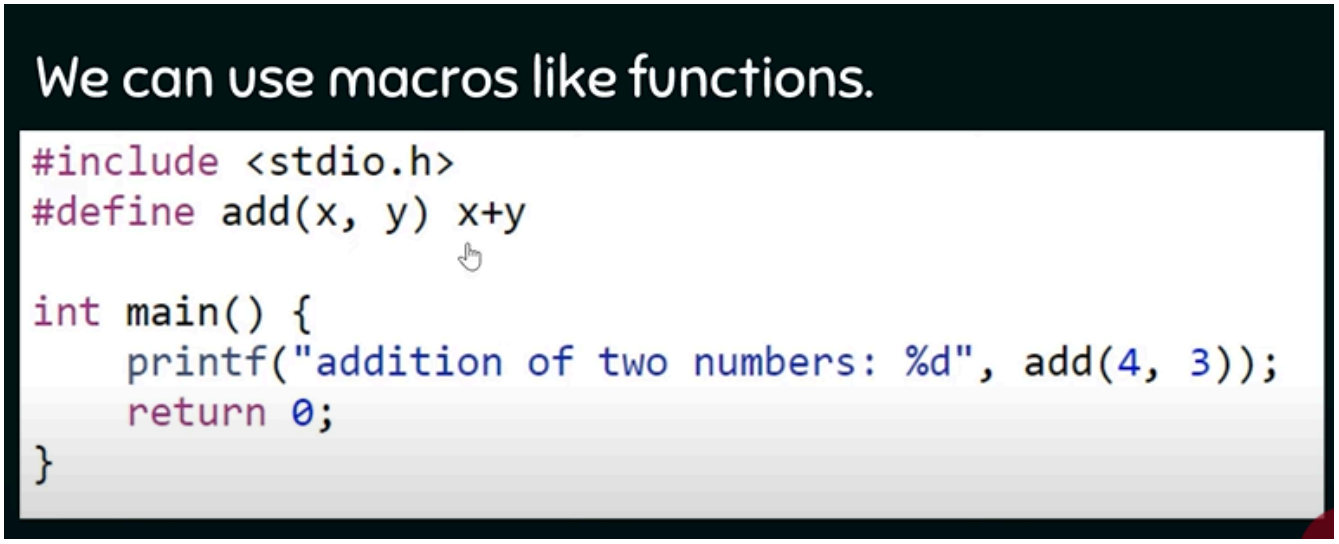
4. static - var is accessible only in this file and can retain its value even after function ends execution

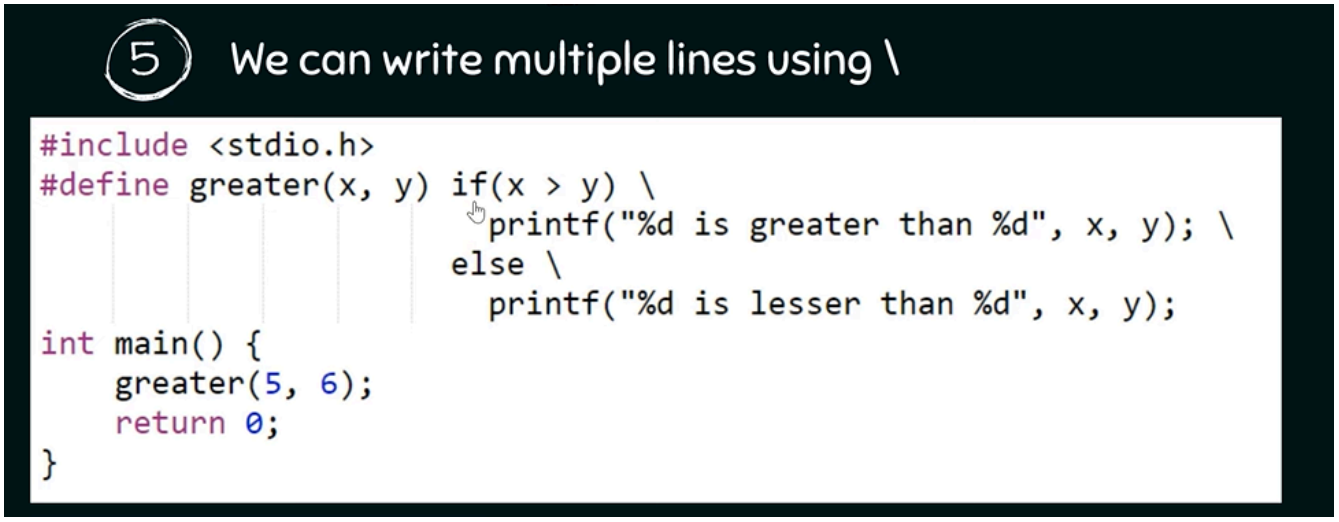
- used to stop modification to the var by some other file.
- By Default, static is initialized to 0.
- static var can only be initialized with a constant value. Initialization with another variable or function not allowed
- can be declared multiple times it will remain in bss segment, but once value is assigned it will stay in Initialized data even if after that we declare again without uninitializing.

- Multiple declaration is allowed but multiple values can't be assigned to same static var in same scope.

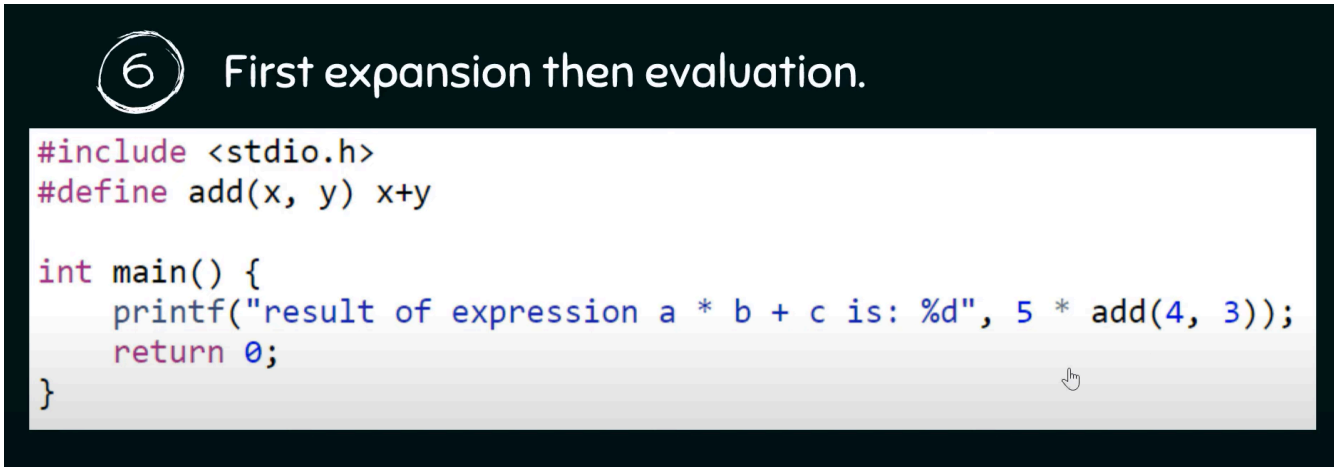
Constants in C -

- Once defined value can't be modified.
- Syntax -
  Globally - `#define PI 3.14159`
  Don't use semi-colon
  OR
  Locally - `const float PI = 3.14159;`
  use semi-colon
- Here, PI name can also be called as Macro.
  Macro -
- Macro can be a var name or function name, which takes up value of expression adjacent to it.





- Macro just replaces expression next to it



  In above Eg. add(4,3) is replace dwith `4+3` and we get overall expression as `5*4+3` , so o/p is 20+3 = 23 and not 5*7 = 35.

- pre-processor replaces name with value adjacent.
- pre defined Macro start with **NAME**
  Eg. - `__DATE__`, `__TIME__`

Memory Layout of C Program-

1. Code segment - where compiled program in form of machine code is stored
2. Data segment - where data/values of our program is stored

- Initialized Data Segment - where variables which have been assigned a value in program are stored.
  It has 2 sections -
  (i) Read only - here initialized constant global variables are stored
  (ii) Read write - here initialized global, extern, static(both local and global) are stored
- Unintialized/ bss segment - where variables which we didn't assigned any value is stored.
  It has 2 sections -
  (i) Read only - here uninitialized constant global variables are stored
  (ii) Read write - here uninitialized global, extern, static(both local and global) are stored
- Stack -
- Heap -
- Cmd-line arg and Env Variables -

Operators in C -

Name of operators	Operators
Arithmetic operators	+ , - , * , / , %
Increment/Decrement operators	++, --
Relational operators	== , ! = , < = , > = , < , >
Logical operators	&& , || , !
Bitwise operators	& , ^ , | , ~ , >> , <<
Assignment operators	= , += , -= , *= , /= , %= , <<= , >>= , &= , ^= , |=
Other operators	?: & * sizeof() ,

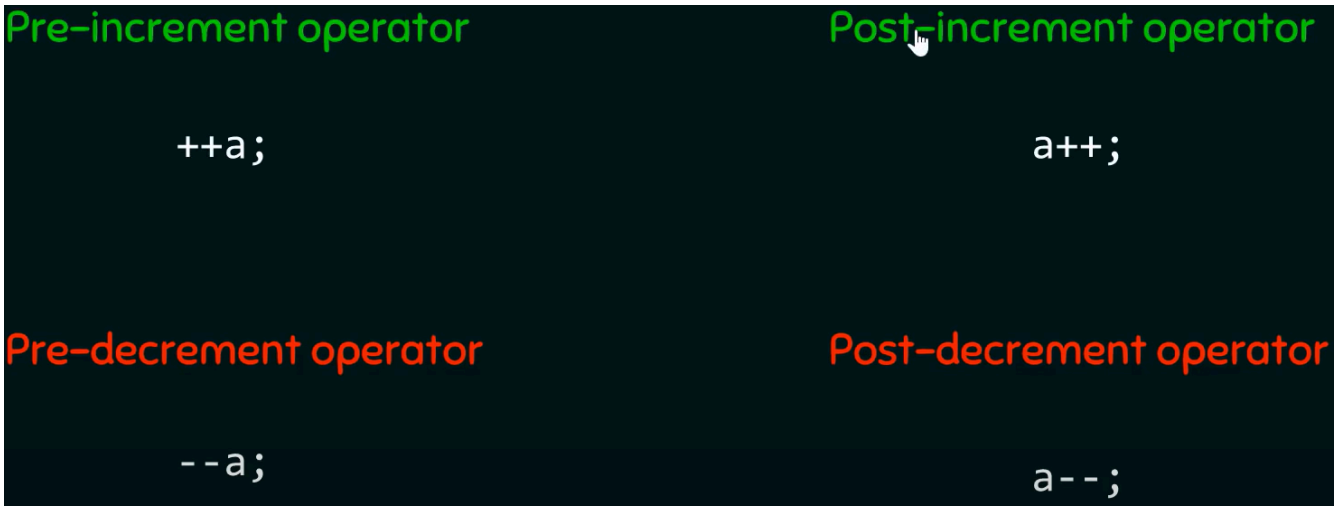
1. Arithmetic Operators - gives a result value as o/p

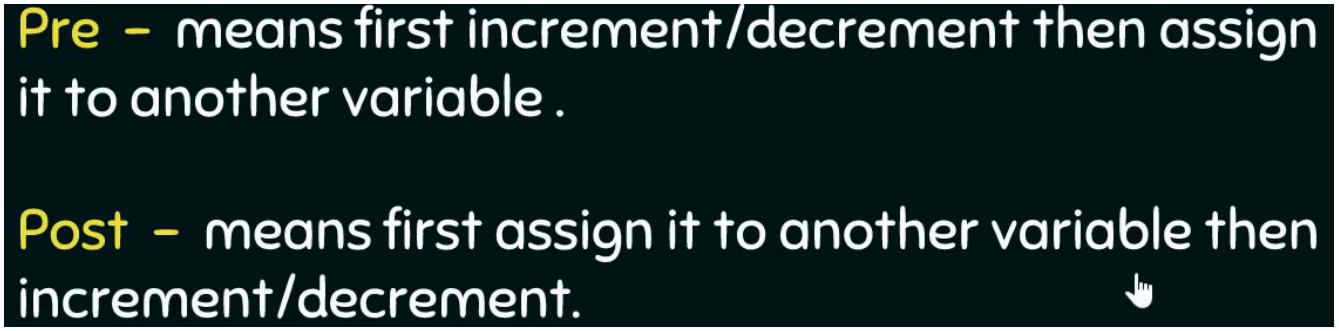
Req. Op - 2

Precedence	Operators	Associativity
↓ Highest	* , / , %	Left to right
Lowest	+ , -	Left to right

2. INC/DEC operators - inc/dec by 1

   Req. Op - 1

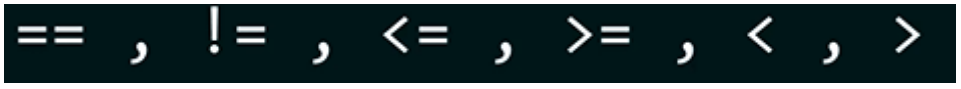




- In Pre-Inc/Dec, substitution is done after the completion of Equation.

3. Relational Operators - returns T(1) or F(0)
   Req. Op - 2

- Used to companre two variables



4. Logical Operators - used to combine 2 or more conditions and Complement any condition

- Returns T or F
-

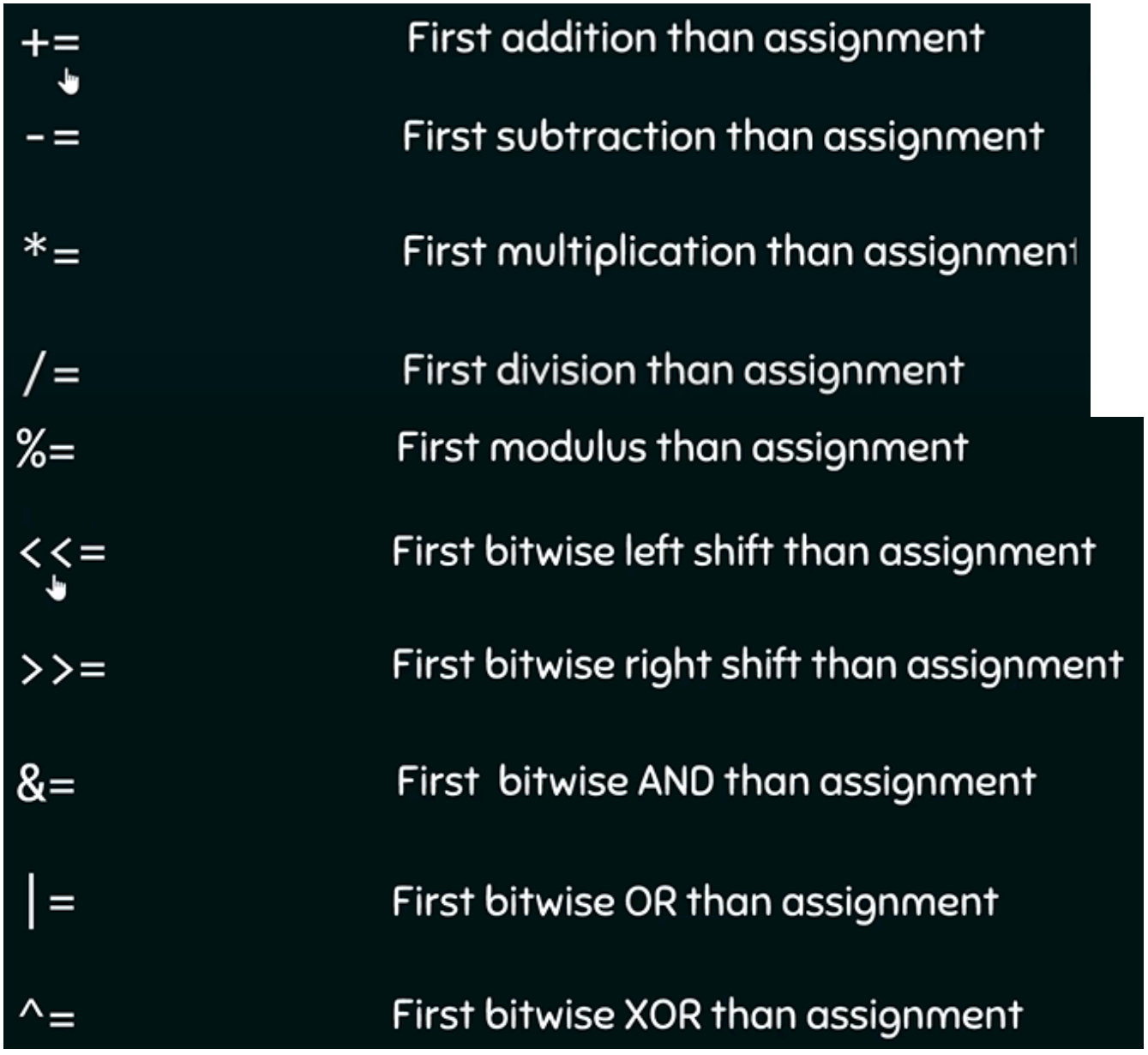
- Short Circuit - rest of condition after that will not be evaluated
  For && - If any one condition is False
  For || - If any one condition is True

5. Bitwise Operators - returns value after bitwise manipulation

- Req. Op - 2

- Left/Right Shift operators -
  Left - var << 2 --> Left shift by 2 bits and fill 0s at ends
  Left shift = var*2^(No. of shifted bits)
  Right - var>>2 __> Right shift by 2 bits and Fill MSBs with 0s
  Right shift = var/2^(No. of shifted bits)
- X-OR operator is used to check if 2 values are same or not

6. Assignment Operator - used to assign values to var
   Req. Op - 2



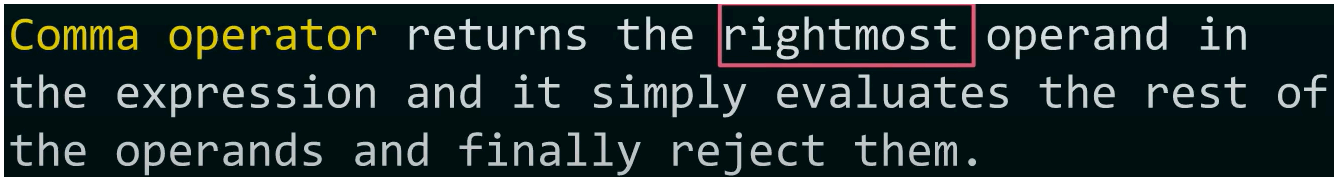
7. Conditional Operator - to replace an simple If-else block
   Req. Op - 3 (the only ternary operator in C)

```
char result;
int marks;

result = (marks > 33) ? 'p' : 'f';
```

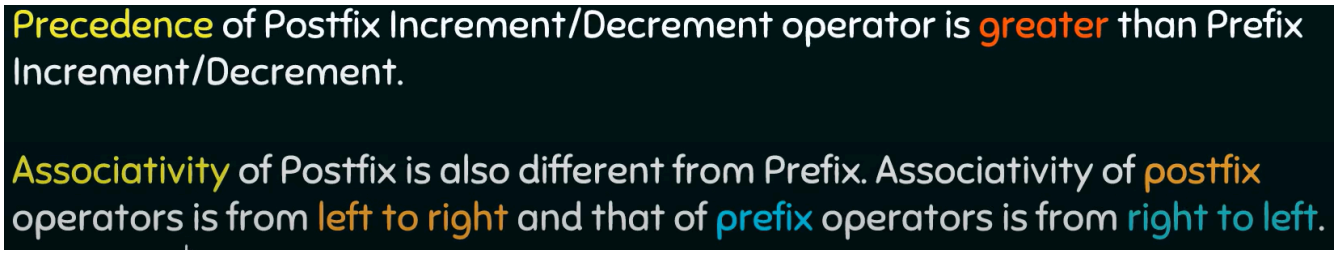
8. Comma Operator - acts as a separator and operator
   separator as in separates 2 var definations
   Opeartor as in Assigns rightmost value to var and Evaluates all expression.

- Have the Least Precedence among all operators



9. Member Access Operators - used to access members in structures.

10. Post/Prefix Inc/Dec operators(++,--) -



- Precedence and Associativity -
  Left to right associativity means in an expression leftside operator will be evaluated first.

Category	Operators	Associativity
Parenthesis/brackets	() [] -> . ++ --	Left to right
Unary	! ~ ++ -- + - * & (type) sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Bitwise Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	= += -= *= /= %= &= ^= |= <<= >>=	Right to left
Comma	,	Left to right

- Here, Postfix operator are in Parenthesis Row and Prefix oprators are in Unary Operators row.
- Only Unary, Conditional, Asignment opearators have associativity from Right to left.
  Rest have associativity from Left to right.
- If we have just one operator between 2 functions, then we can't say which function will be called first. Behaviour is compiler dependent.

```
a = fun1() + fun2();
```

Output is not defined -

```
Output:    NesoAcademy2
             OR
           AcademyNeso2
```

Technical Terms -

1. Pangram - A pangram is a sentence containing every letter in the English Alphabet.
2. Palindrome - the reverse of the string is the same as the string(Eg - madam)
3. Re-entrant functions - These are functions whose execution can be stopped in between to go serve an interrupt and then resume again without hampering its previous actions.
4. Lexicographically - means in the order of words appearing in the dictionary