UNIT-1

Introduction to C

History of C:

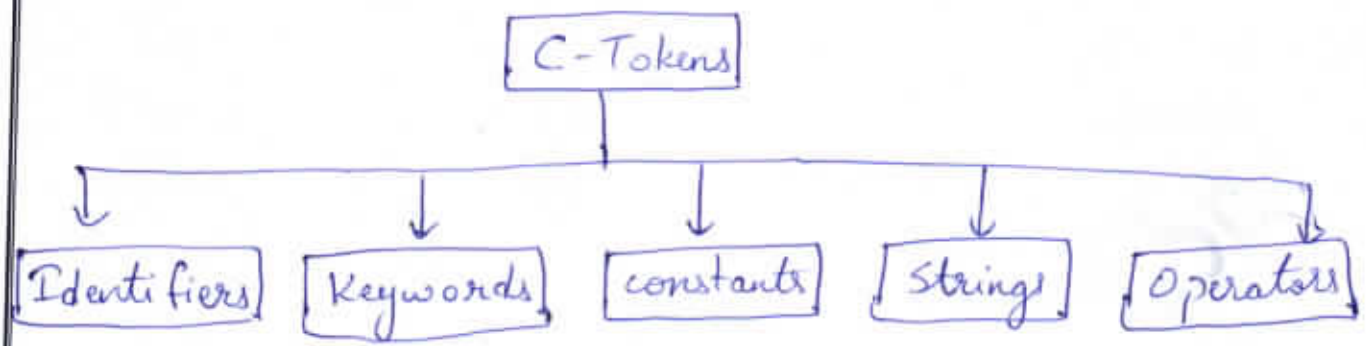
1960 -	Algol 60	- By International committee
1963 -	CPL (Computer Programming language)	- By Cambridge University.
1967 -	BCPL (Basic Computer Programming language)	- By Cambridge University.
1970 -	B	- By Ken Thompson at AT & T bell laboratories.
1972 -	C	- By Dennis Ritche of AT & T Bel laboratories

Features of C:

* It is a middle level language as it contains the features of both high level language & low level language.
* It is a structured programming language.
* It is case sensitive.
* It supports data structures such as arrays, functions, pointers, structures & unions.
* It is highly portable.
* It is a kind of modular programming.
* It is used for developing applications & system softwares.
* It has some library functions & inbuilt functions.

C-Tokens

These are smallest individual units which represent a single character or a group of characters which has a specific meaning.



Identifiers:

These are user defined names that are given to the various program elements such as variables, functions, arrays, structures etc.

Limitations:

* It allows alphabets, digits & the special character underscore (_).
* Both upper case & lower case letters are permitted.
* Upper case & lower case can't be interchanged.
* An identifier name should not start with a digit.
* Underscore is rarely used.
* Keywords should not be used as identifiers.

Ex: Max_Price , _total - valid identifiers

Min area , 'a' , "college" - invalid identifiers.

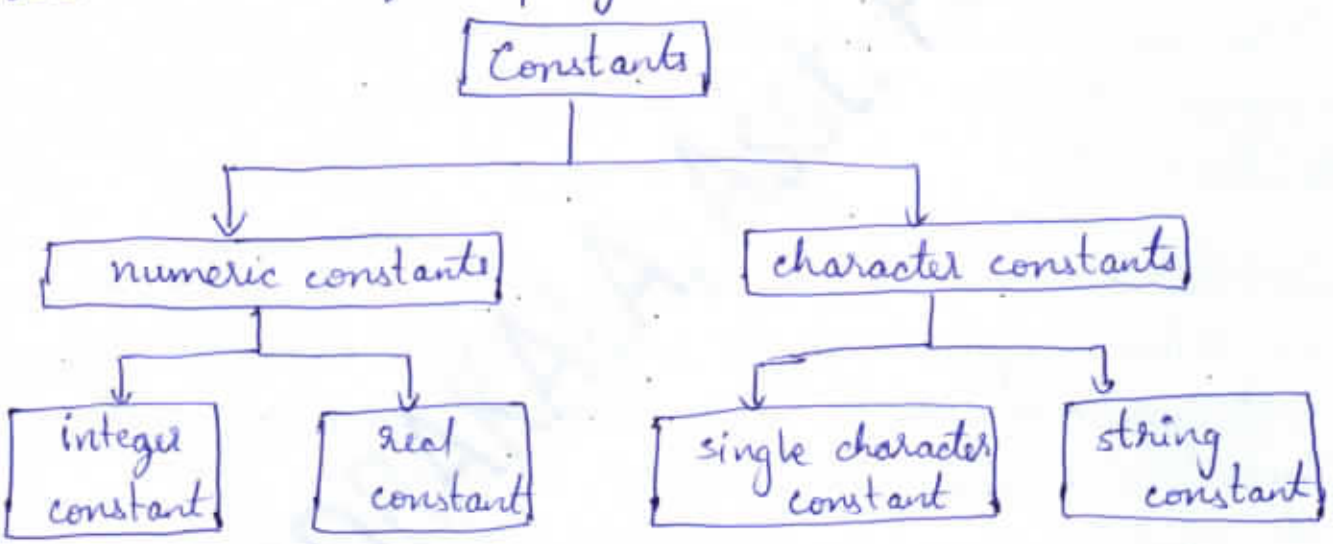
Keywords:

* These are reserved words which have standard & pre-defined meaning in 'C'.
* These are used only for their intended purpose.

Ex:	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

Constants:-

Constants are fixed values that donot change during the execution of a program.



Integer constants : It refers to a sequence of digits.

It is of 3 types:

① Decimal integers : It consists of set of digits, 0 through 9, preceded by an optional - or + sign.

Ex: 123, -321, 0, 654321, +78. → valid

② Embedded spaces, commas, & non-digit characters are not permitted between digits.

Ex: 15 750, 20,000, $1000 → not valid.

② Octal constants: It consists of any combination of digits from the set 0 through 7, with a leading 0.

Ex: 037, 0, 0435, 0551

③ Hexadecimal constants: A sequence of digits preceded by 0x and 0X.

They may include alphabets A through F or a through f. A through F represents numbers 10 through 15.

Ex: $0X_2$, 0x9F, 0xbcd, 0x

Real Constants:

These are shown in decimal notation, having a whole number followed by a decimal point & fractional part.

Ex: 0.0083, −0.75, +247.0, 215., .95, +.5, −.71

Single Character Constants:

* It contains a single character enclosed in a pair of single quotes.

Ex: '5', 'X', ';', ' '

* Character constants have integer values called ASCII values.

Ex: printf ("%d", 'a'); o/p = 97

printf ("%c", 97); o/p = a

String constants:

* A string constant is a sequence of characters enclosed in double quotes.
* Characters may be numbers, letters, special characters & blank space.

Ex: "Hello!", "1987", "WELL DONE", '5+3', "X"

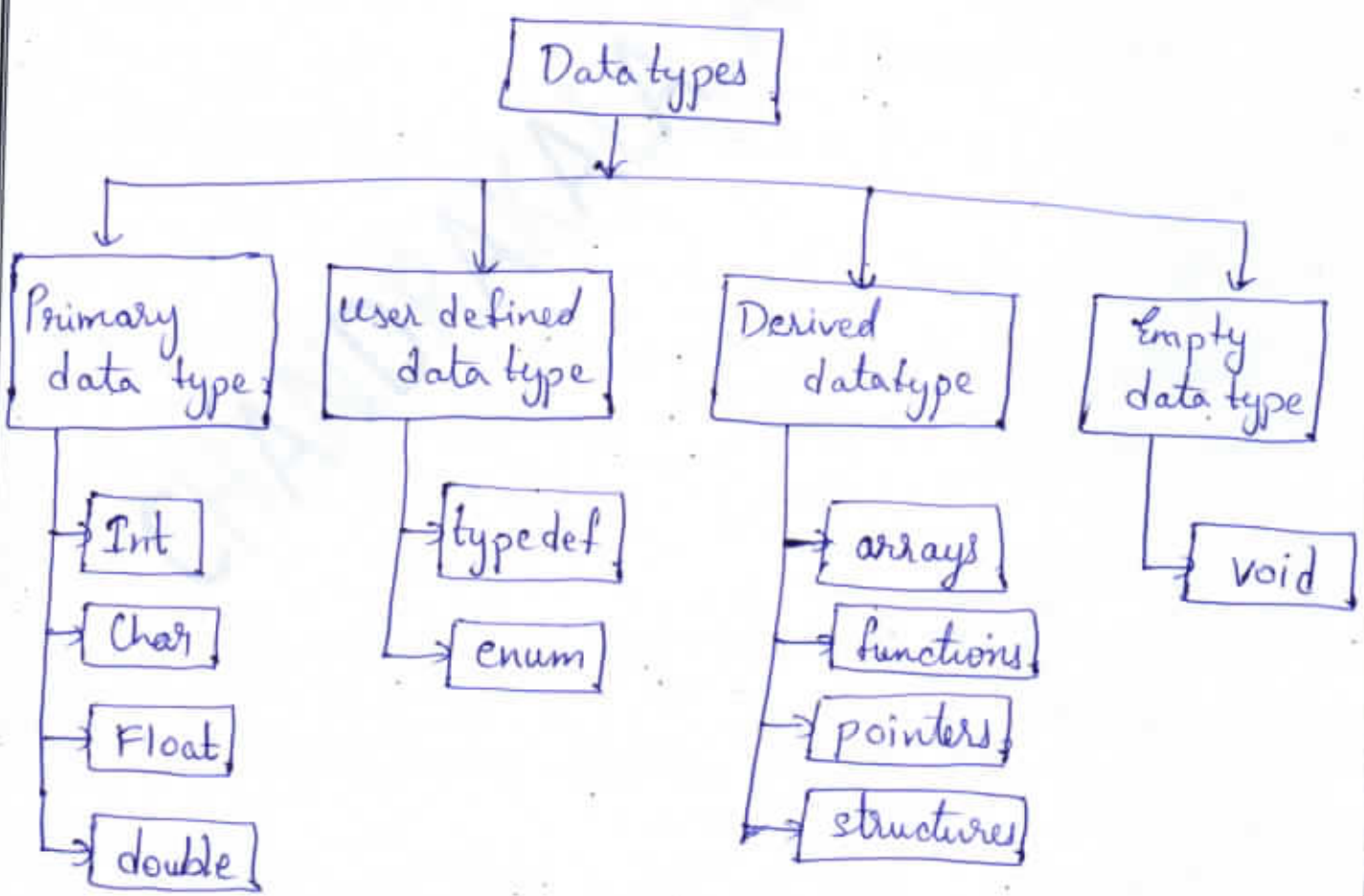
Backslash character constants:

* These are used in output functions.
* Each represent one character.
* These character combinations are called as escape sequences.

Ex:
- '\n' – newline
- '\t' – horizontal tab
- '\v' – vertical tab
- '\b' – backspace.
- '\0' – null

Data type :-

It specifies type of data, size of data & range of data.



① int:

Description: Integer type is used to store positive & negative integer.

format: %d

storage space: 2 bytes

range: -32,768 to 32,767

② char:

Description: A character denotes any alphabet, digit or special symbol represented in single quotes. (Stores a single character)

format: %c

storage space: 1 byte

range: -128 to 127

③ float:

Description: It is used to store real number, with single precision floating point number. (Precision of 6 digits after decimal points).

format: %f

storage space: 4 bytes

range: 3.4 e-38 to 3.4 e+38

④ double:

Description: It stores real numbers with double precision. (Precision of 14 digits after decimal points).

format: %lf

storage space: 8 bytes

range: 1.7 e-308 to 1.7 e+308

Void: It is a special datatype used for:

* to specify that a function doesn't returns any value.
* to specify a function takes no arguments. Ex: ① void print(void) ② void *ptr
* to create generic pointers.

Variable:

A variable is an identifier that is used to represent a single data item.

Rules for constructing a variable:

* A variable name is a combination of alphabets, digits and underscore.
* Other than the underscore no special characters is allowed.
* The first character in the variable name must be an alphabet.
* The variable name should not be of keyword.

Declaration of variables:

Syntax: `datatype v1, v2, ..... vn;`

where $v_1, v_2 \ldots v_n$ are names of variables.

Assigning values to variables:

We can assign values to variables in two ways.

`variable_name = constant ;`

`datatype variable_name = constant;`

Ex1:

```
main()
{
int m1, m2, m3;
m1 = 10;
m2 = 20;
m3 = 30;
printf("%d %d %d", m1, m2, m3);
}
```

output

10 20 30

Ex2:
```
main()
{
int m1=10, m2=20, m3=30;
printf("%d %d %d", m1, m2, m3);
}
```

output
10 20 30

Ex3:
```
main()
{
char name;
name = "Computer";
printf("%s", name);
}
```

output
computer

Ex4:
```
main()
{
char name = "Computer";
printf("%s", name);
}
```

output
computer

Operators:

An operator is a symbol that tells a computer to perform a particular task.

The types of operators are:

(i) Arithmetic operators
(ii) Relational operators
(iii) logical operators
(iv) Assignment operators.
(v) Increment & decrement operators.
(vi) Bitwise operators
(vii) Conditional operators
(viii) Special operators.

Arithmetic operators:

These are basic & common operations performed in computer language.

Operator	Meaning	Example
+	Addition or unary plus	a+b
-	Subtraction or unary minus	a-b
*	Multiplication	a*b
/	Division	a/b
%	Modulo division	a%b

There are 3 types of integer arithmetic, real arithmetic and mixed arithmetic.

(i) integer arithmetic:

The operation is performed only for integers & the result should also be an integer.

Ex) a = 14 , b = 4

a+b = 18
a-b = 10
a*b = 56
a/b = 3 (quotient without decimals)
a%b = 2 (remainder)

(ii) Real arithmetic: It is performed only for real operands. The operator % cannot be used with real operands.

Ex: x = 6.0 / 7.0 = 0.857143
y = 1.0 / 3.0 = 0.333333
z = -2.0 / 3.0 = -0.666667

(iii) Mixed-mode arithmetic: Here one of the operand is real and the other is integer.

Ex: 15 / 10.0 = 1.5

where as

15 / 10 = 1

Relational operators:

These are used to perform comparison between two values. These operators returns TRUE (1) if the comparison condition is true otherwise FALSE (0).

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

Ex: 4.5 <= 10 TRUE (1)

-35 >= 0 FALSE (0)

10 < 7+5 TRUE (1)

Logical Operators

When we want to take a decision basing on two or more conditions then we use logical operators.

Operator	Meaning
&&	logical AND
||	logical OR
!	logical NOT

Logical AND (&&)

Truth table

A	B	A&&B
T	T	T
T	F	F
F	T	F
F	F	F

logical OR (||)

Truth table

A	B	A || B
T	T	T
T	F	T
F	T	T
F	F	F

Logical NOT (!)

Truth table

A	!A
T	F
F	T

Ex: 8>6 && 6<10 TRUE (1)

9>4 || 7<3 TRUE (1)

8!=8 FALSE (0)

Assignment operators

This operator is used to assign a constant value or the result of an expression to a variable.

operator	Meaning
=	Assigns RHS value to LHS value

Shorthand assignment operator:

```
v  op  =  exp;
```

where, v = variable
op = operator
exp = expression
= is shorthand assignment operator

is same as

```
V = V op exp;
```

Simple assignment operators	Shorthand assignment operators
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

Increment & decrement Operators:

The operator	Meaning	Example
++	adds 1 to the operand	++m or m++
--	subtracts 1 from operand	--m or m--

++m is equal to m = m+1
--m is equal to m = m-1

But ++m and m++ behave differently.

Ex1: m = 5;
y = ++m;
Ans: m = 6
y = 6

Ex2: m = 5;
y = m++;
Ans: y = 5
m = 6

It is similar when we use ++ operator & -- operator in subscripted variables.

* The above points are applicable to decrement operator also.

Conditional Operator :-

```
exp1 ? exp2 : exp3;
```

Here ternary operator pair "?:" is used.
where exp1, exp2, exp3 are expressions.

* If exp1 is true, exp2 becomes value of expression.
* If exp1 is false, exp3 becomes value of expression.

Ex: a=10;
b=15;
x=(a>b)? a:b;

Ans: a>b
10>15 (false)
So, x=b
x=15

Bitwise operator :-

* These are used to manipulate data at bit level.
* These are not applicable for float or double datatype

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right
~	One's complement

Bitwise AND :

It needs '1' in both numbers

```
12 - 00001100
25 - 00011001
&  ----------
     00001000 = 8
```

Bitwise OR :

Atleast of the numbers is '1'

```
12 - 00001100
25 - 00011001
|  ----------
     00011101 = 29
```

Bitwise exclusive OR:

'0' if same numbers &
'1' if different numbers

```
12-  00001100
25-  00011001
^    00010101 = 21
```

Bitwise 1's complement:

0's becomes 1's &
1's becomes 0's.

```
12-00001100
~  11110011 - 243
```

Shift left:

It is used for left shifting.

```
12-00001100
<<1-00011000 - 24
<<2-00110000 - 48
```

Shift right:

It is used for right shifting.

```
12- 00001100
>>1-00000110 - 6
>>2-00000011 - 3
```

Special operators:

(i) Comma operator: It is used to link the related expressions together.

Ex: `a = (x = 10, y = 5, x+y);`

It assigns 10 to x, 5 to y and 10+5 to x+y.
finally a = 15.

(ii) sizeof operator: It returns number of bytes the operand occupies in memory.
It is a compile time operator.

Syntax: `sizeof (operand);`

Ex: `y = sizeof(int);` here value of y = 2 bytes.
`z = sizeof(a)` here value depends on datatype of a

Evaluation of Expressions:

Expressions are evaluated using an assignment statement.

variable = expression;

Algebraic expression	C Expression
$a \times b - c$	a * b - c
$(m+n)(x+y)$	(m+n) * (x+y)
$\frac{ab}{c}$	a * b / c
$3x^2+2x+1$	3 * x * x + 2 * x + 1
$\frac{x}{y}+c$	x / y + c

Precedence of arithmetic operators:

An arithmetic expression without parenthesis will be evaluated from left to right using rules of precedence of operators.

High priority * / %

low priority + -

Ex1: x = 9 - 12/3 + 3 * 2 - 1

x = 9 - 4 + 3 * 2 - 1

x = 9 - 4 + 6 - 1

x = 5 + 6 - 1

x = 11 - 1

x = 10

Ex2: x = 9 - 12/(3+3) * (2-1)

x = 9 - 12 / 6 * (2-1)

x = 9 - 12 / 6 * 1

x = 9 - 2 * 1

x = 9 - 2

x = 7

Ex3: x = 9 - (12 / (3+3) * 2) - 1

= 9 - (12 / 6 * 2) - 1

= 9 - (2 * 2) - 1

= 9 - 4 - 1

= 9 - 5

= 4

Ex4: x = 9 - ((12 / 3) + 3 * 2) - 1

= 9 - (6 + 3 * 2) - 1

= 9 - (6 + 6) - 1

= 9 - 12 - 1

= 9 - 11

= -2

Operator Precedence & Associativity

The operators of same precedence are evaluated either from left to right or from right to left, depending on the level. This is known as associativity property of an operator.

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

Here addition operator has higher priority than logical operator (&&) and relational operator (== and <)

if (x == 25 && y < 10)

Here x == 25 is FALSE (0)
y < 10 is TRUE (1)

(Assume x values 20, y value 5)

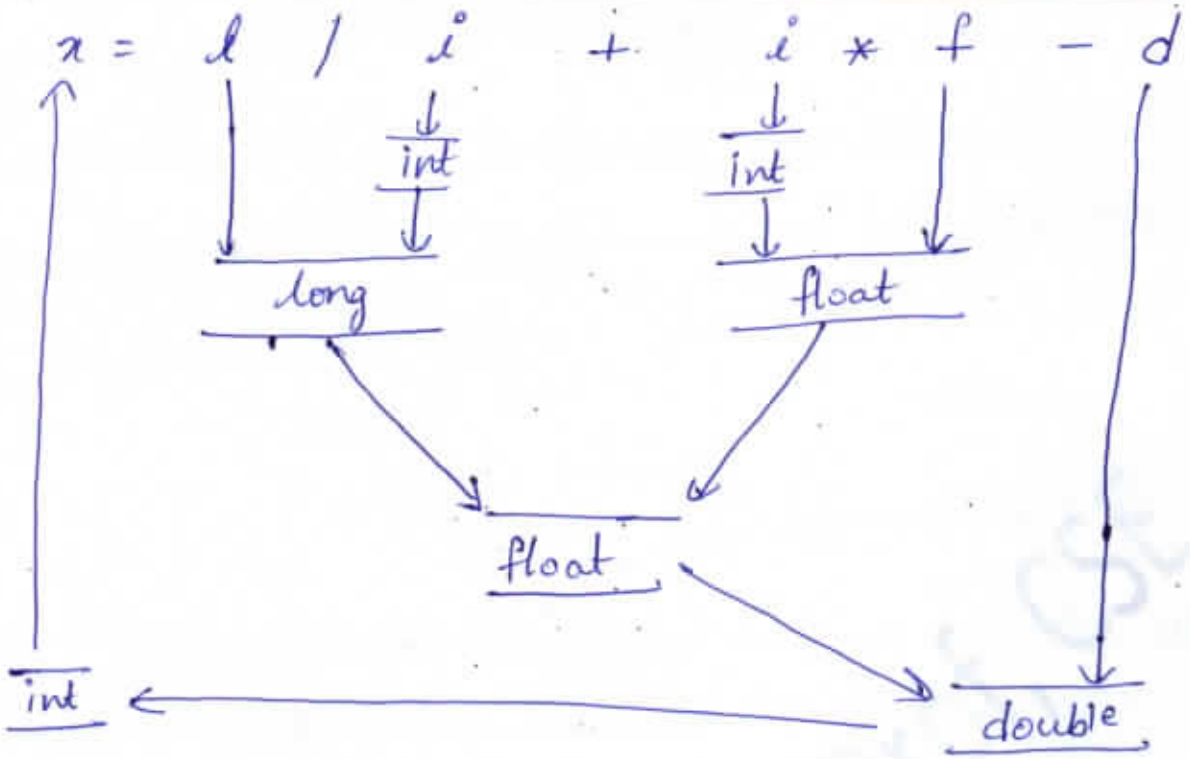
Finally if (FALSE && TRUE)
Ans is FALSE

Type Conversions in expressions

Automatic type conversion:

If operands are of different types, then the 'lower' type is converted to the 'higher' type.

Ex:
```
int i, x;
float f;
double d;
long int l;
```



Casting a value:

Syntax: (type name) expression

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 & result is 5.
b = (double) sum/n	Division is done in floating point mode.
y = (int) (a+b)	Result of a+b is converted to integer.
z = (int) a+b	a is converted to integer & then added to b.
p = cos((double) x)	Converts x to double before using it.