x) String constant:

- The string constant are the sequence of character enclosed within double quotation marks.

Assignment: const char a = "Hello"

Note: (i): "a" = string constant → assigns value equal to ASCII

(ii) : 'a' = single character → string constant not equal to ASCII

When compiler reads string constant, it reads the first character and adds zero at the end indicating the end of string.

Eg: "Jenny" $\xrightarrow{\text{compiles}}$ "Jenny\0"

5 6 characters.

C-operators:

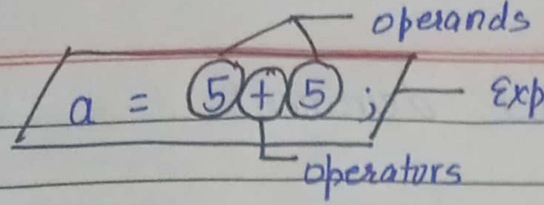
x) Operands: The value on which operators does work are called operands.

x) Operators: The symbols that directs the compiler on what manipulation is to be done to a data are called operators.

x) Expression: The sequence of operators and operands which gives single value after processing is called expression.

Date. No.

Eg:



⇒Types of Operators Based on operands:

a): Unary operators:

The operators that does operation on only one operand is called unary operators.

The unary operators are as follows:

i) Unary minus ⇒ −
ii) Unary increment ⇒ ++
iii) Unary decrement ⇒ --
iv) Logical NOT ⇒ !
v) Address of ⇒ &
vi) size of (type)

(i): Unary minus:

It changes the sign of the value.

Eg:
```
int a,b;
b = 10;
a = 5;
c= a+(-b);   ⇒ - ~~10~~ 5
a=-b ;       => -10
```

(ii): Unary increment and decrement:

Unary increment sign: ++
Unary decrement sign: --

These operators are of two types: ~~post~~ prefix and postfi

Syntax of prefix:
++ X
-- X

Here,
first operation is done on X
and then ~~assi~~ implement.

Eg: int y = 10, x = 11;
y = X ++ ;
Here,
firstly, y = X
then, x = x+1
~~pri~~ printf ("%d", y) ⇒ 11
printf ("%d", x) ⇒ 12

Syntax of postfix:
X++
X--

Here,
first implementation is done
and then operation is done
on X.

Eg: int x = 11, y = 10;
y = ++ X ;
Here,
firstly, x = x + 1
then, y = x
printf ("%d", y) ⇒ 12
printf ("%d", x) ⇒ 12

Similarly, the same concept is applied for decrement.

X++ / ++ X demonstrates X = X+1;
X-- / -- X demonstrates X = X-1;

(iii) Logical NOT::

Logical NOT sign: !

→ This sign reverses the logical state of any operand.

```
Eg: int c, d;
~~Eg~~: int x = 11, y = 10;
    c = ! (x>y)   ⇒ returns true
    d = ! (~~x~~y>x)   ⇒ returns false
   printf ("%d", c); ⇒ 1
   printf ("%d", d); ⇒ 0
```

(iv) Address of:

Symbol: &

It retrives the address of any operands from memory.

It is used in points and scanf function.

(v): sizeof

Size of gives the memory of any datatype or variable in ~~bytes~~ bytes.

b): Binary operators:

The operators that does operatoion on two operands are called binary operators.

The binary operators are as follows:

Arithmetic, relational, Logical, Bitwise, Equality, Comma operator, Assignment operator.

→ These are operators based on operation.

<c>: Ternary Operators:

- Ternary operators are the operators that require three operands to perform operation. It is also called conditional operators.

Syntax: expression 1 ? expression 2 : expression 3

Here,

- expression 1 is condition.
- expression 2 is implemented if exp-1 is true and expression 3 is implemented if exp-1 is false.

- This operator is used in the place of if-else.

Eg:
```
int a=10, b=15;
int x;
x = (a>b) ? a : b;
printf ("%d", x);  => 15
```

=> Types of Operators Based on Operations:

The types of operators based on operations are as follows:

(i): Arithmetic operators:
(ii): Assignment operators:
(iii) Increment & decrement:
(iv) Logical operators
(v) Relational operators
(vi) Bitwise operators
(vii) Special operators.

(i): Arithmetic operators:

Arithmetic operators are as follows:

+ => addition => gives sum
− => subtraction => gives difference
* => product => gives product
/ => division => gives quotient
% => modulo => gives remainder.

We can use +, −, *, / for all positive or negative integer values.

While operating % with negative integer, the sign of the remainder provided is the same as the first operand.

Eg: −10 % 7 = −3
10 % −7 = 3

Here,

operator precedence : * / % — ①
+ − — ②

operator associativity : Left → Right.

(ii) Assignment operators:

Assignment operators assign value to a variable.

Eg: a = 5;

it has associativity from right to left.

Here, left hand side must be variable
right hand side must be int, char or float.

o) Short hand operators:

if,

a = a+1;   then,   a += 1;

b = b-2   then,   b -= 2;

Here, short hand operators are used when both variables are on ~~the~~ ~~sa~~ both of the sides.

(iii): Increment and Decrement operator:

(* explained before in unary operator)

→ used with both int and float value.

Eg:
```
#include <stdio.h>
void main ()
{
int a=5, b, c, d;
 b= ++a;
 c= a++;
d= ++a;
printf ("%d", a);
}
```

Here, the steps are,

a = a+1   so,

~~b=~~ b=a   a = 8

c = a   b = 6

a = a+1   c = 6

~~d=~~ a = a+1   d = 8

d = a

(iv) Relational operators:

- The operators that compares the relationship between two operands are called relational operators.
- Relational operators are also called comparison operators and is used in decision making.
- It returns Boolean value 1 ⇒ true
0 ⇒ false.

Operators:

>	⇒ greater than
<	⇒ less than
<=	⇒ less than or equal to
>=	⇒ greater than or equal to
==	⇒ equals to
!=	⇒ not equal to.

- We use relational operators to compare int, float and char.
- float values are generally not used. to maintain precisi
- characters are compared by using their ASCII values.

Syntax: Arithmetic Exp. 1 Relational operator Arithmetic Exp. 2

Format ~~speci~~ specifier: %d ⇒ as relational operators give integer value ie, 0 & 1.

Associativity: Left → Right

Eg:
```
# include <stdio.h>
# include <conio.h>
void main ()
{
    int a=18,b=9;
    clrscr();
    printf ("%d", a<b);                  => 0
    printf ("%d"; ~~'b' < 'b'~~ 'c' > 'b');     => 1
    getch();
}
```

Eg:
```
# include <stdio.h>
# include <conio.h>
void main ()
{
  int a=18, b=9, c,d,e=10;
  clrscr();
  c = b++ ;
  d=b;
  printf ("%d", a< b<c>d);         => 0
  printf ("%d", b = = e );         => 1
  printf ("%d", c+1 >e) ;          => 0
  printf ("%d", a+c == b >e< c+d);  => 0
  getch();
}
```

Rough:
a=18
b=10
c=9
d=10
e=10

18+9 == 10 > 10 < 9+10
27 == 10> 10< 19
27== 0 <19. 1 =>0

Precedence : <, >, >=, <= — (1)
== — (2)

Arithmetic operators is implement ahead of relational operators.

Eg:
```
#include <stdio.h>
    void main ()
    {
     int  a=18, b=9, c, d, e=10, f;
     c = b++
     d = b;
     f = a>b>d<c
     printf ("%d", f!=1); => 0
     }
     printf ("%d") a+c==b>=e < c+d != 1); => 1
```

a = 18
b = 10
c = 9
d = 10
e = 10

f = $\underline{18 > 10} > 10 < 9$
$\underline{1 > 10} < 9$
$0 < 9 = 1$

f! = 1 → 0

18+9 == 10 >= 10 < 9+10 != 1
27 == $\underline{10 >= 10}$ < 19 != 1
27 == 1 < 9 != 1
$\underline{27 == 1}$!= 1
0 != 1 → True.

(v) Logical Operator:
 The operators used to test more than one condition are called logical operators.

The operators are: && => AND
|| => OR
! => NOT

- Logical operators returns value true (1) or false (0).

Syntax:

Relational expression 1	Logical operator	Relational expression 2

↑ logical expression.

Date. No.

→ Logical AND:

Symbol: &&

It only returns true value if all the conditions are true.

→ AND process proceeds only if given value comes true.

Eg:

if a && b then, b is executed.
(1)

if a && b then, b is not executed.
(0)

→ Logical OR:

Symbol: ||

It returns ~~true~~ false value if all the conditions are false

→ OR process proceeds only if given value comes false.

Eg:

if a || b , b is executed
(0)

if a || b , b is not executed.
(1)

Note: Any other value other than zero is considered true.

→ Logical NOT:

Symbol: !

It only works on one operand and it negates the value of the operands.

Eg: 5! = 0

0! = 1

Precedence: ! , &&, ||

Eg: (i):
```
void main ()
{
int a=10, b=5, result;
printf ("%d", ~~a&&b~~ 4 && !0! ); => 1
printf ("%d", 4 && 0); => 0
result = (a>b) && a++
printf ("%d ; result); => 1
}
```

(ii):
```
~~m~~ void main ().
{
int a=4, b=6, result;
result = a>b && printf("Jenny") || printf("lectures") ||
                                            printf("Jk");
printf ("%d", result); => 1
}
```

a>b => 0&&X || printf T || X

Here, the value of result is 1.
Also,
during result, a>b is false so printf ("Jenny") is not executed
and since printf ("lectures") is true, printf ("Jk") is not implemented.

Output:
lectures 1

Date. No.

(vi): Bitwise Operator:

- Bit: It is the smallest level in computer memory to store data.

- The operators that perform operation at bit level.

Operators:

i) Bitwise AND: &
ii) Bitwise OR: |
iii) Bitwise XOR; ^
iv) Bitwise NOT ~
v) Bitwise left shift <<
vi) Bitwise right shift >>

→ Bitwise AND:
Symbol: &

Eg:
```
void main ( )
{
   int a = 11, b = 5, c ;
   c = a & b.
   printf ("%d", c); => 1
}
```

8	4	2	1
0	1	0	1
1	0	1	1
0	0	0	1

Here, every bit in c gives 1 iff the both corresponding value on a & b is 1.

→ Bitwise OR:

Symbol: |

```
8 4 2 1
1 0 1 0
0 1 0 1
-------
1 1 1 1 =15
```

Eg: int a=10, b=5, c;
c = a | b ⇒ 15

Here, every bit in C gives 1 if any one of the corresponding value is 1 in a or b.

→ Bitwise XOR

Symbol: ^

```
8 4 2 1
1 0 1 0
0 1 1 0
-------
1 1 0 0 =12
```

~~Here~~ Eg: int a=10, b=6 &, c;
c = a ^ b ⇒ 12

Here, every bit in C gives 1 if any one of the corresponding value is 1 and gives 0 if both of the corresponding value in ~~a~~ a and b is same.

```
1 0 1 0
0 1 1 0
-------
0 0 1 0
```

```
Eg: void main()
    {
     int a=10, b=6;
     printf ("%d", a&b); => 2
     printf ("%d", a|b); => 11
     printf ("%d", a^b); => 12
     printf ("%d", a&b && b+1 || 0 || b++) => 1
     printf ("%d", b); => 6
    }
```

→ Bitwise left shift:

Symbol: <<

syntax: variable << units.

Eg: a<<2,

shift variable a value by 2 bits. towards left.

Eg: int a=10; 00001010

c = a<<2; ⇒00101000

printf("%d", c); ⇒ 40

Shortcut. value of c = variable value × $2^{\text{shifting unit}}$

Eg: $c = 10 \times 2^2$

$\therefore c = 40$

Trailing bits is filled with zero.

→ Bitwise right shift:

Symbol: >>

Syntax: variable >> unit.

Eg: a >> 2.

shift variable a value by 2 bits towards right.

Eg: int a=10;

c= a>>2; 00001010

printf("%d",c); ⇒ 2 ⇒00000010 → 2

leading bits is filled with 0

Shortcut value of c = $\frac{\text{variable value}}{2^{\text{unit value.}}}$ Eg: $\frac{10}{2^2} = 2$

- Bitwise NOT :

Symbol : ~

It inverts the value in bits.

ie, 1 ⟶ 0 , 0 ⟶ 1

Eg: int a = 5; 0 1 0 1

b = ~a; → 1 0 1 0 = 10

printf ("%d", b); ⇒ 10 .

(vii): Comma operator :

→ it is an special operator.

Associativity: Left ⟶ Right

It has the least precendence.

Functioning: First expression is evaluated and its value is rejected, the second expression is evaluated and its result is returned.

Eg: i) int a = 5, 4; Here, a = 5.

ii) int a = (5, 4); Here, a = 4.

(iii): int a;

a = (printf ("Jenny"), 2);

Here,

a = 2

output : Jenny.

Operators Precedence and Associativity:

Precedence order	Operators	Associativity.
1	( ) . → ++ -- (post)	L → R
2	++ --(prefix) + - ! ~ * & sizeof {These are unary operators	R → L
3	* / % {binary}	L → R
4	+ - {binary}	L → R
5	<< >> {bitwise}	L → R
6	< <= > >= {relational)	L → R
7	== != (equality)	L → R
8	& (Bitwise AND)	L → R
9	^ (Bitwise XOR)	L → R
10	| (Bitwise OR)	L → R
11	&& (Logical AND)	L → R
12	|| (Logical OR)	L → R

13	? : (Ternary)	L→R
14	= += -= /= %= *= ^= >>= <<=	R→L
15	,	L→R.