Page No.
Date :

UNIT - 2

* OPERATORS :-
  - It is the symbol that performs an operati
  - Operators acts on the variables which are known as operands

Eg :- a + b
a, b are operands & + is an operator

* TYPES OF OPERATOR :-

1. ARITHMETIC OPERATORS :-

	OPERATOR	NAME	USES
1.	+	Add	Sum of two data
2.	-	Subtract	Subtract two data
3.	*	Multiply	product of two data
4.	/	Division	gives Quotient of two data
5.	%	Modulo	gives remainder of two data.

- % Modulo operator operates only on integer data.

Page No.
Date :

```
# include <stdio.h>
 Void main ()
{
   int a = 9 , b = 4 , c ;

   c = a+b ;
   printf (" a+b = %d \n" , c ) ;

   c = a-b ;
   printf ("a - b = % d \n" , c) ;

   c = a* b ;
   printf (" a*b = %d \n" , c ) ;

   c = a/b ;
   printf (" a/b = % d \n" , c ) ;

   c = a % b ;
   printf (" Remainder when a divided by b =
          %d \n" , c ) ;

   }
```

- OUTPUT

```
a + b = 13
a - b = 5
a * b = 36
a / b = 2
Remainder when a divided by b = 1
```

Page No.
Date :

```
#include <stdio.h>
void main ( )
{
    int  a,b ;
printf ("\n Input two integer data : ") ;
scanf ("%d %d", &a, &b);

printf ("\n sum = %d", a+b);
printf ("\n subtract = %d", a-b);

printf ("\n product = %d", a*b);

printf ("\n division = %d", a/b);

printf ("\n remainder = %d", a%b);

}
```

- OUTPUT :

```
Input two integer data :
 14
 8
sum = 22
Subtraction = 6
product = 112
division = 1
remainder = 6
```

Page No.
Date :

2. RELATIONAL OPERATOR :-

- It checks the relationship between two operands.
- It is used to compare numerical data.

OPERATOR	MEANING
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	equal to
!=	not equal to.

- If the relation is true, it returns 1.
  If relation is false, it returns value 0.

```
# include <stdio.h>
int main ()
{
    int m = 40, n = 20;

    if (m == n)
    {
        printf ("m & n are equal");

    }
else
{
    printf ("m & n are not equal");

}
}
```

Page No.
Date :

. OUTPUT
m & n are not equal.

```
#include <stdio.h>
int main ()
{
    int a=5, b=5, c=10;
printf ("%d == %d = %d \n", a, b, a==b);
                                // True.
printf ("%d > %d = %d \n", a, c, a>c);
                                // False
printf ("%d < %d = %d \n", a, c, a<c);
                                // True.
printf ("%d != %d = %d \n", a, b, a!=b);
                                // False.
printf ("%d >= %d = %d \n", a, c, a>=c);
                                //false
printf ("%d <= %d = %d \n", a, c, a<=c);
                                // true.
return 0;
}
```

• OUTPUT.
5 == 5 = 1

Page No.
Date :

5 > 10 = 0
5 < 10 = 1
5 != 5 = 0
5 >= 10 = 0
5 <= 10 = 1

Page No.
Date :

3. LOGICAL OPERATORS :-

- It is used to Combine the results of two or more conditions.
- An expression Containing logical operator returns either 0 or 1 depending upon whether expression results are false or true.

OPERATOR	MEANING
&&	Logical AND
||	Logical OR
!	Logical NOT

- LOGICAL AND (&&) :- If any one cond^n is false then complete cond^n becomes false.

TRUTH TABLE

OP 1	OP 2	OP 1 && OP 2
T	T	T (1)
T	F	F (0)
F	T	F (0)
F	F	F (0)

- LOGICAL OR (||) :- If any one cond^n is true then complete cond^n becomes true.

TRUTH TABLE

OP 1	OP 2	OP 1 || OP 2
T	T	T
T	F	T
F	T	T
F	F	F

Page No.
Date :

- LOGICAL NOT (!) :- It reverses the value of expression it operates on.

TRUTH TABLE

OP 1	OP 1 !
T	F
F	T

```
# include <stdio.h>
int main ()
{
   int a = 5, b = 5, c = 10, result;

   result = (a = b) && (c > b);   // True
printf ("(a = b) && (c > b) equals to %d \n", result);

   result = (a = b) || (c < b);   // True.
printf ("(a = b) || (c < b) equals to %d \n", result);

   result = ! (a = = b);   // false.
printf (" ! (a = = b) equals to %d \n", result);

   return 0;

}
```

OUTPUT.

(a = b) && (c > b) equals to 1
(a = b) || (c < b) equals to 1
! (a = = b) equals to 0.

Page No.
Date :

4. ASSIGNMENT OPERATOR :

- They are used to assign a value or an expression or value of a variable to another variable.

- SYNTAX :-

Variable name = value (or) Expression (or) variable.

- Eg :-
```
x = 10 ;
y = a+b ;
z = p ;
```

-

OPERATOR	MEANING
+=	x = x+y
-=	x = x-y
*=	x = x*y
/=	x = x/y
%=	x = x%y

```
# include <stdio.h>
int main ()
{
    int a = 5, c ;
    c = a ;
printf ("c = %d \n", c) ;

    c += a ; // c = c+a
printf ("c = %d \n", c) ;
```

```
c -= a;  // c = c - a
printf ("c = %d \n", c);

c *= a;  // c = c x a
printf (" c = %d \n", c);

c /= a;  // c = c/a
printf ("c.= %d \n", c);

c %= a;  // c = c % a
printf ("c = %d \n", c);

return 0;
}
```

OUTPUT

```
c = 5
c = 10
c = 5   0
c = 25
c = 5   1
c = 0
```

*

```
# include <stdio.h>
  Void main ()
{
   int a, b;
    a = 10;
     b = a;
printf ("\n a = %d \t b = %d", a, b);
}
```

- OUTPUT :- a = 10 b = 10

Page No.
Date :

5. INCREMENT & DECREMENT OPERATOR :.

- It is used to increement or decreement the value of given variable by 1.

- We can only use these operators with variables & not with expressions or constants.

- Eg:

```
int a=1, b=1;

++b;   // valid
++3;   // invalid - (Constant value 3)
++(a+b); // invalid (expression a+b)
```

★ PREFIX INCREMENT OPERATOR :-

- It increements the current value of the available variable immediately & only then this value is used in an expression.

- SYNTAX :- ++ variable ;

Eg:

```
# include <stdio.h>
# include <conio.h>

Void main( )
{
    int P, Q;
    Q = 10;
    P = ++Q;
  printf ("P: %d", P);
```

```
printf ("q : %d", q);
getch ();
}
```

OUTPUT :-

P : 11
q : 11

* PREFIX DECREMENT OPERATOR :-

```
# include <stdio.h>
# include <conio.h>
 void main ()
{
   int p, q;
   q = 10;
   p = --q;

printf ("P : %d", p);
printf ("q : %d", q);

getch ();
}
```

- OUTPUT

P : 9
q : 9

* POSTFIX INCREEMENT OPERATOR :-

- The value of the variable is first used

Page No.
Date :

inside the provided expression, & then its value gets increemented by 1.

- SYNTAX :- Variable ++;

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int a, b;
    b = 10;
    a = b++;

 printf ("a : %d", a);
 printf ("b : %d", b);

   getch ();
}
```

- OUTPUT

a : 10
b : 11

* POSTFIX DECREEMENT OPERATOR :-

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int a, b;
     b = 10;
     a = b--;
```

```
printf ("a : %d", a);
printf ("b : %d", b);
getch ();
}
```

OUTPUT:

```
a : 10
b : 9
```

Page No.
Date:

* CONDITIONAL OPERATORS: (TERNARY OPERATOR)

- It checks the condition & executes the statement depending on condition
- It is ternary operator i.e it works on 3 operands.
- It consist of 2 symbols:
  : Question mark (?)
  : Colon (:).
- SYNTAX: (Condition ? exp 1 : exp 2);
- It first evaluate the condition, if it is true then "exp 1" is evaluated, if condⁿ is false then "exp 2" is evaluated.

```
# include <stdio.h>
  int main ()
{
    int x = 1, Y ;

    Y = (x == 1 ? 2 : 0);

   printf ("x value is %d \n", x);

   printf ("y value is %d", Y );

}
```

OUTPUT :
x value is 1
y value is 2

Page No.
Date :

```
# include <stdio.h>
  int main ()
{
    char February ;
    int days ;

 printf ("If this year is Leap year, enter 1 .
            If not enter any integer : ") ;
scanf (" %c", & February ) ;
days = (February == '1') ? 29 : 28 ;

 printf (" Number of days in february = %d",
                    days ) ;

return 0 ;

}
```

OUTPUT

If this year is leap year, enter 1 . If not enter any integer : 1
Number of days in February = 29 .

Page No.
Date:

* BITWISE OPERATORS :-

- They are used to manipulate data at bit level
- It operates on integers only.
- It may not be applied to float.

OPERATOR	MEANING
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	One's complement

* BITWISE AND OPERATOR : [&]

- The o/p of bitwise AND is 1, if corresponding bits of two operands is 1. If either bit of an operand is zero the result of corresponding bit is zero.

- Eg:- Bitwise AND operation of 12 & 25

```
  12 =  0000 1100
  25 =  0001 1001
  ---------------
        0000 1000 = 8   (In decimal)
```

```
# include <stdio.h>
int main ()
{
   int a=12, b=25;
printf ("output = %d", a & b);
   return 0;
}
```

Page No.
Date:

• OUTPUT:
output = 8

* BITWISE OR OPERATOR: (|)
- o/p is 1 if atleast one corresponding bit of two operands is 1.

```
# include <stdio.h>
  int main ( )
  {
     int a = 12, b = 25;
     printf ("output = %d", a|b);
      return 0;

  }
```

• OUTPUT
output = 29

* BITWISE XOR (exclusive OR) OPERATOR :- (^)
- The result of bitwise XOR operator is 1 if corresponding bits of two operands are opposite.

```
# include <stdio.h>
  int main ( )
{
    int a = 12, b = 25;
   printf ("Output = %d", a^b);
     return 0;
}
```

• OUTPUT :-
output = 21

Page No.
Date :

* BITWISE COMPLEMENT OPERATOR (~) :-
- It is an unary operator i.e works on only one operand.
- It changes 1 to 0 & 0 to 1
- Eg :- ~ 00100011 = 11011100

```
# include <stdio.h>
int main()
{
printf("complement = %d\n", ~35);
printf("complement = %d\n", ~-12);

  return 0;
}
```

- OUTPUT

```
complement = -36
Complement = 11
```

* BITWISE LEFT SHIFT OPERATOR [<<]
- It shifts all bits towards left by certain number of specified bits.

* BITWISE RIGHT SHIFT OPERATOR [>>]
- It shifts all bits towards right by certain number of specified bits.
- Shifts the number in binary form by one place in the operation & returns the result.

Page No.
Date:

* SPECIAL OPERATORS :-

1. COMMA OPERATOR :.

- It is used to separate statement elements such as variables, constants or expressions & is used to link the related expressions together, such expressions can be evaluated from left to right & value of right most expressions is value of combined expression.

- Eg :- Val (a = 3, b = 9, c = 77, a+c)
First assigns the value 3 to a, then assigns 9 to b, then assigns 77 to c & finally assigns 80 to c.

2. SIZE OF OPERATOR :-

- It is a unary operator that returns the length in bytes of the specified variable & also useful to find bytes occupied by specified variable in the memory.

- SYNTAX :. sizeof (variable_name);
- Eg :. sizeof(a);

```
# include <stdio.h>
  int main ()
{
   int  a, e [10];
   float b;
   double c;
```

```
char d;
printf ("Size of int = %lu bytes \n", Size
                    of (a) );
printf ("size of float = % lu bytes \n", size
                    of (b) );
printf ("size of double = % lu bytes \n",
                    size of (c) );
printf ("size of char = % lu bytes \n",
                    size of (d));

printf ("size of integer type array having
          10 elements = % lu bytes \n",
          size of (e) );

return 0;

}
```

- OUTPUT

```
Size of int = 4 bytes.
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
Size of integer type array having 10
 elements = 40 bytes.
```

Page No.
Date :

* EXPRESSIONS :

- It is a combination of operators & operands which reduces to a single value.
- An operator indicates an operation to be performed on data that yields a value.
- An operand is the data item on which an operation is performed.
- A simple expression consist of only one operator.

Eg :- 3 + 5 simple expression
6 + 8 * 7 complex expression.
a + b * x
4 * a * c / b
a * x * x + b * x + c

* OPERATOR PRECEDENCE :

- Arithematic operators are evaluated left to right using precedence of operator when expression is written without parenthesis.

- Arithematic expression evaluation is carried out using two phases from left to right.

1. FIRST PHASE : Highest priority operator (* / %) are evaluated in 1st phase

2. SECOND PHASE : The Lowest priority operator (+ −) are evaluated in 2nd phase.

- Whenever parenthesis are used, the expression within parenthesis gets highest priority. If 2 or more set of parenthesis

occur one after another then the expression contained in left most set is evaluated first & right most in the last.

Eg:- $9 - 12/(3+3) * (2-1)$

→

1st phase :- $9 - 12/6 * (2-1)$

$9 - 12/6 * 1$

2nd phase $9 - 2 * 1$

$9 - 2$

3rd phase 7

* RULES FOR EVALUATION OF EXPRESSION :-
- Evaluate the sub-expression from left to right if parenthesized.
- Evaluate the arithemetic expression from left to right using rules of precedence.
- The Highest precedence is given to a expression within parenthesis.
- Apply the associative rule, if more operators of same precedence occurs.

* OPERATOR PRECEDENCE & ASSOCIATIVITY :-

- Complex expressions are executed according to precedence of operators
- Associativity specifies the order in which the operators are evaluated with same preeedence in a complex expression.

Page No.
Date:

- Associativity is of two ways i.e left to right & right to left.
- Left to right associativity evaluates an expression starting from left & moving towards right.
- Right to left associativity proceeds from right to left.

	OPERATOR	DESCRIPTION	PRECEDENCE	ASSOCIATIVITY
1.	( )	Function call	1	Left to right
	[ ]	Square brackets		
2.	+, -, ++, --, !!, ~, *, &, sizeof	Unary +, unary -, Increement, decrement, Not operator, Complement, pointer operator, Adress operator, sizeof operator.	2	Right to left.
3.	*, /, %	Multiplication, Division, Modulo	3	Left to right.
4.	+	Addition	4	Left to right.
	-	Subtraction		
5.	<<	Left shift	5	Left to right.
	>>	right shift		
6.	<, <=, >, >=	Relational operator	6	Left to right.

Page No.
Date :

	OPERATOR	DESCRIPTION	PRECEDENCE	ASSOCIATIVITY
7.	==	Equality	7	Left to right.
	!=	Inequality		
8.	&	Bitwise AND	8	Left to right.
9.	^	Bitwise XOR	9	Left to right
10.	|	Bitwise OR	10	Left to right
11.	&&	Logical AND	11	Left to right
12.	||	Logical OR	12	Left to right
13.	? :	Conditional	13	Right to left
14.	=, *=, /=, %=, +=, -=, &=, ^=, <<=, >>=	Assignment operator	14	Right to left
15.	,	Comma operator	15	Left to right.