

CE143: COMPUTER CONCEPTS & PROGRAMMING

UNIT-3 Operators and Expression in 'C'

N. A. Shaikh

nishatshaikh.it@charusat.ac.in

Topics to be covered

- Classification of Operators
- Unary, Binary and Ternary Operators
- Arithmetic expression
- Evaluation of expression
- Type conversion
 - Implicit
 - Explicit
- Precedence and Associativity
- Various library functions from math.h

Operators

- The symbols which are used to perform logical and mathematical operations are called **C operators**.

TYPES OF C OPERATORS:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment/decrement operators
6. Conditional operators (ternary operators)
7. Bit wise operators
8. Special operators

Arithmetic operators

Used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus

Arithmetic Operators/Operation	Example
+ (Addition)	$A+B$
- (Subtraction)	$A-B$
* (multiplication)	$A*B$
/ (Division)	A/B
% (Modulus)	$A\%B$

Here,

a and b are variables which are known as **operands**.

Arithmetic operators

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

```
void main()
```

```
{
```

```
    int a,b, add,sub,mul,div,mod;
```

```
    printf("Enter the value of a:");
```

```
    scanf("%d",&a);
```

```
    printf("Enter the value of b:");
```

```
    scanf("%d",&b);
```

```
    add = a+b;
```

```
    sub = a-b;
```

```
    mul = a*b;
```

```
    div = a/b;
```

```
    mod = a%b;
```

```
    printf("Addition of a & b is : %d\n", add);
```

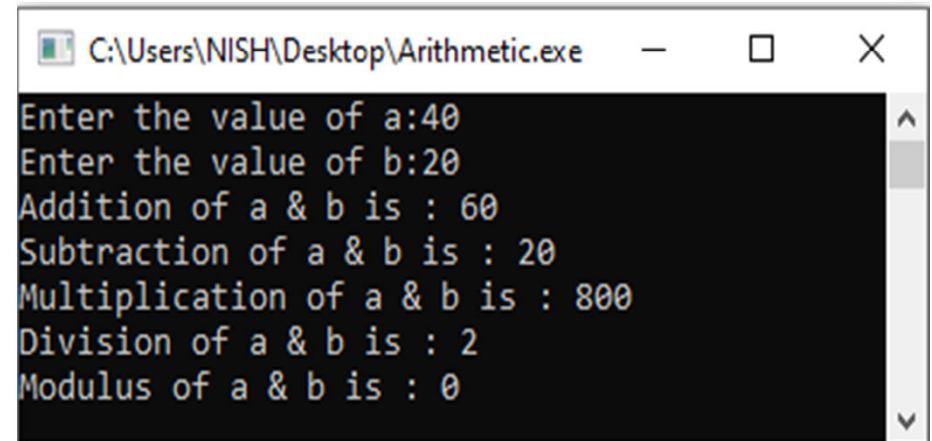
```
    printf("Subtraction of a & b is : %d\n", sub);
```

```
    printf("Multiplication of a & b is : %d\n", mul);
```

```
    printf("Division of a & b is : %d\n", div);
```

```
    printf("Modulus of a & b is : %d\n", mod);
```

```
}
```



```
C:\Users\NISH\Desktop\Arithmetic.exe
Enter the value of a:40
Enter the value of b:20
Addition of a & b is : 60
Subtraction of a & b is : 20
Multiplication of a & b is : 800
Division of a & b is : 2
Modulus of a & b is : 0
```

Arithmetic operators

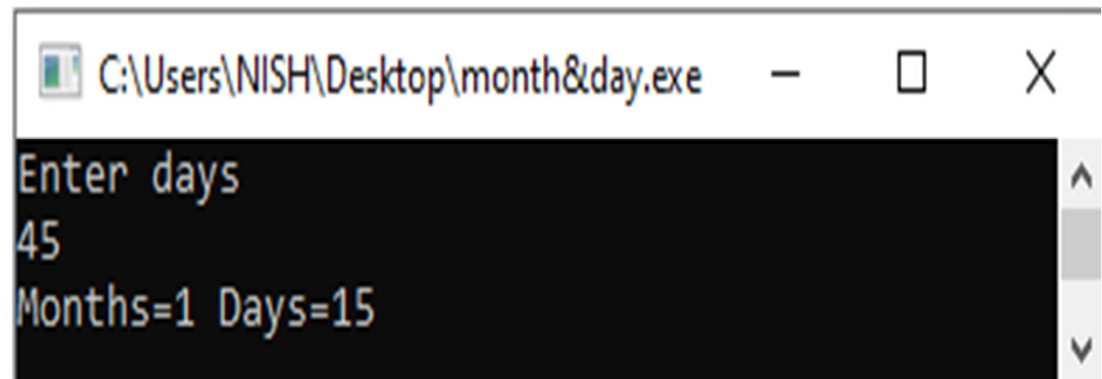
C Program to convert a given number of days into months and days

```
#include<stdio.h>

main()
{
    int months, days;

    printf("Enter days\n");
    scanf("%d", &days);

    months=days/30;
    days=days%30;
    printf("Months=%d Days=%d", months, days);
}
```



```
C:\Users\NISH\Desktop\month&day.exe
Enter days
45
Months=1 Days=15
```

Relational operators

Used to find the relation between two variables.
i.e. to compare the values of two variables in a C program.

Operators	Example/Description
>	<code>x > y</code> (x is greater than y)
<	<code>x < y</code> (x is less than y)
>=	<code>x >= y</code> (x is greater than or equal to y)
<=	<code>x <= y</code> (x is less than or equal to y)
==	<code>x == y</code> (x is equal to y)
!=	<code>x != y</code> (x is not equal to y)

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

Relational operators are used in **decision making** and **loops**

Relational operators

```
#include <stdio.h>

int main()
{
    int a = 5, b = 5;

    printf("a>b: %d \n", a > b);
    printf("a<b: %d \n", a < b);
    printf("a>=b: %d \n", a >= b);
    printf("a<=b: %d \n", a <= b);
    printf("a==b: %d \n", a == b);
    printf("a!=b: %d \n", a != b);

    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\NISH\Desktop\relational.exe". The output of the program is displayed as follows:

```
a<b: 0
a>=b: 1
a<=b: 1
a==b: 1
a!=b: 0
```


Logical operators

Used to perform logical operations on the given expressions

Operators	Example/Description
&& (logical AND)	<code>(x>5)&&(y<5)</code> It returns true when both conditions are true
(logical OR)	<code>(x>=10) (y>=10)</code> It returns true when at-least one of the condition is true
! (logical NOT)	<code>!((x>5)&&(y<5))</code> It reverses the state of the operand “ <code>((x>5) && (y<5))</code> ” If “ <code>((x>5) && (y<5))</code> ” is true, logical NOT operator makes it false

commonly used in **decision making**

Logical operators

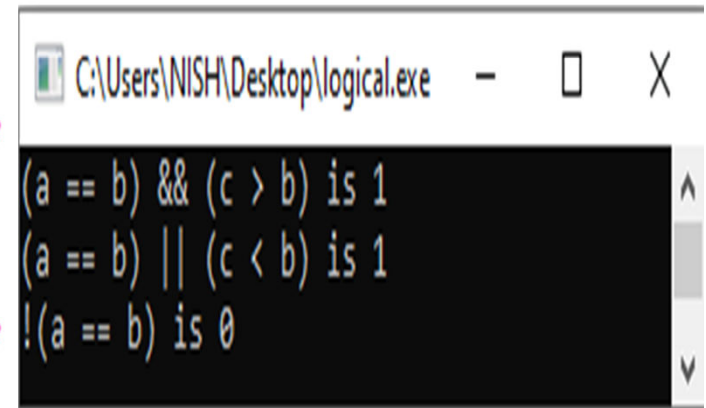
```
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

A screenshot of a Windows command prompt window titled "C:\Users\NISH\Desktop\logical.exe". The window has a black background and white text. It displays the output of the C program: "(a == b) && (c > b) is 1", "(a == b) || (c < b) is 1", and "!(a == b) is 0". The output is aligned to the left, matching the printf statements in the code. The window includes standard Windows window controls (minimize, maximize, close) in the title bar.

```
C:\Users\NISH\Desktop\logical.exe
(a == b) && (c > b) is 1
(a == b) || (c < b) is 1
!(a == b) is 0
```

Logical operators



Practical-3.2: Any year is input through the keyboard. Write a program to determine whether the year is a leap year or not. Use the logical operators `&&` and `||`.

Assignment operators

values for the variables are assigned using assignment operators.

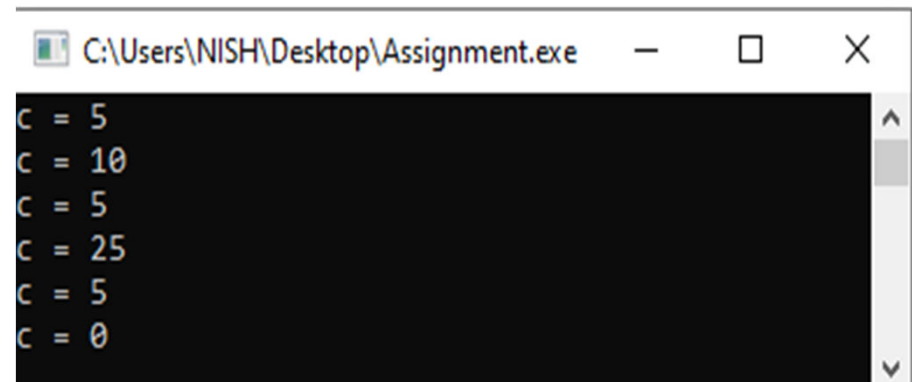
Operators	Example/Description
=	<code>sum = 10;</code> 10 is assigned to variable sum
+=	<code>sum += 10;</code> This is same as <code>sum = sum + 10</code>
-=	<code>sum -= 10;</code> This is same as <code>sum = sum - 10</code>
*=	<code>sum *= 10;</code> This is same as <code>sum = sum * 10</code>
/=	<code>sum /= 10;</code> This is same as <code>sum = sum / 10</code>
%=	<code>sum %= 10;</code> This is same as <code>sum = sum % 10</code>
&=	<code>sum&=10;</code> This is same as <code>sum = sum & 10</code>
^=	<code>sum ^= 10;</code> This is same as <code>sum = sum ^ 10</code>

Assignment operators

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

    c = a;           // c is 5
    printf("c = %d\n", c);
    c += a;          // c is 10
    printf("c = %d\n", c);
    c -= a;          // c is 5
    printf("c = %d\n", c);
    c *= a;          // c is 25
    printf("c = %d\n", c);
    c /= a;          // c is 5
    printf("c = %d\n", c);
    c %= a;          // c = 0
    printf("c = %d\n", c);

    return 0;
}
```



```
C:\Users\NISH\Desktop\Assignment.exe
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

Increment operators

Increment operators(++):

used to increase the value of the variable by one

Syntax:

++var_name; (PRE INCREMENT /PREFIX INCREMENT)

var_name++; (POST INCREMENT /POSTFIX INCREMENT)

Example:

++ i ;

i ++ ;

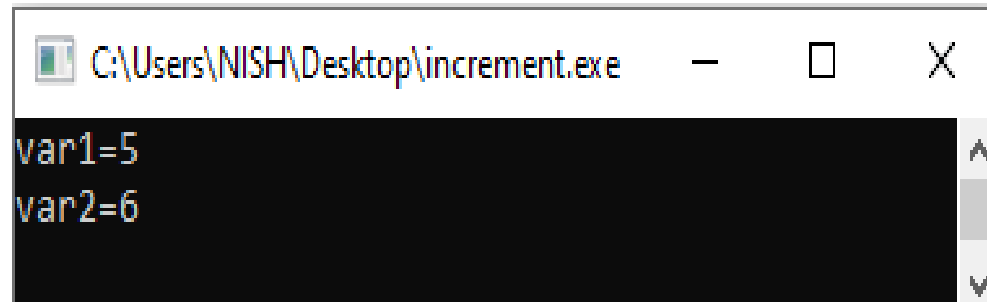
Increment operators

```
#include <stdio.h>

int main()
{
    int var1 = 5, var2 = 5;

    printf("var1=%d\n", var1++); //var1 is displayed then increased to 6.
    printf("var2=%d\n", ++var2); // var2 is increased to 6 then displayed.

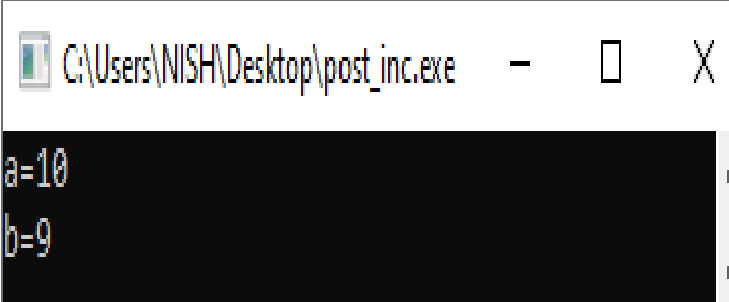
    return 0;
}
```



Increment operators

Postfix Increment

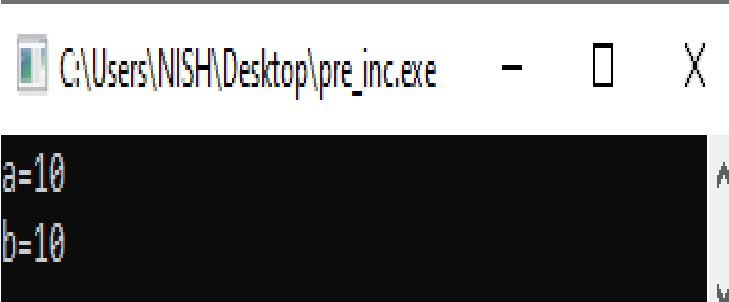
```
#include<stdio.h>
void main()
{
    int a=9,b;
    b=a++;
    printf("a=%d\n",a);
    printf("b=%d",b);
}
```



```
C:\Users\NISH\Desktop\post_inc.exe
a=10
b=9
```

Prefix Increment

```
#include<stdio.h>
void main()
{
    int a=9,b;
    b=++a;
    printf("a=%d\n",a);
    printf("b=%d",b);
}
```



```
C:\Users\NISH\Desktop\pre_inc.exe
a=10
b=10
```


Decrement operators

Decrement operators(--):

used to decrease the value of the variable by one

Syntax:

--var_name; (PRE DECREMENT /PREFIX DECREMENT)

var_name--; (POST DECREMENT /POSTFIX DECREMENT)

Example:

```
-- i;  
i --;
```

Decrement operators

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

```
int main()
```

```
{
```

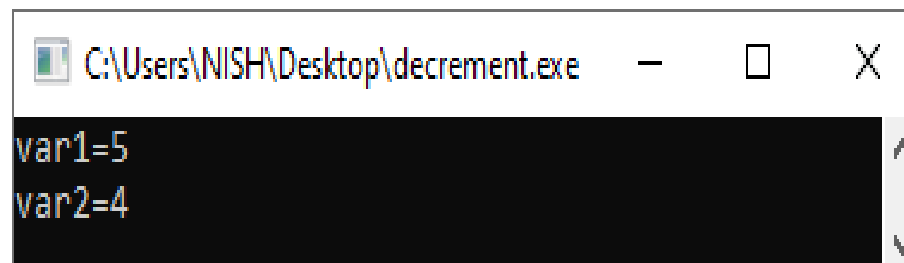
```
    int var1 = 5, var2 = 5;
```

```
    printf("var1=%d\n", var1--); //var1 is displayed then decremented to 4.
```

```
    printf("var2=%d\n", --var2); // var2 is decreased to 4 then displayed.
```

```
    return 0;
```

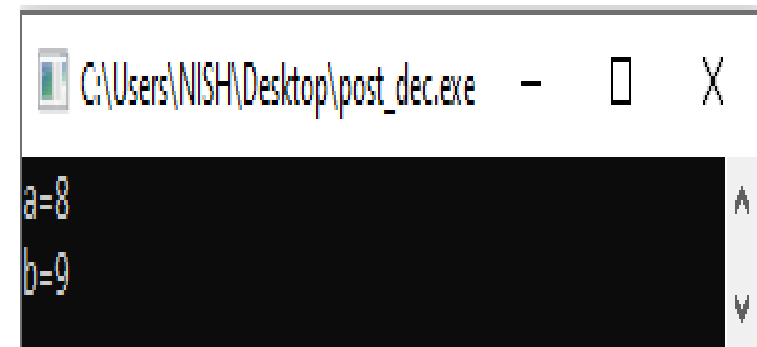
```
}
```



Decrement operators

Postfix Decrement

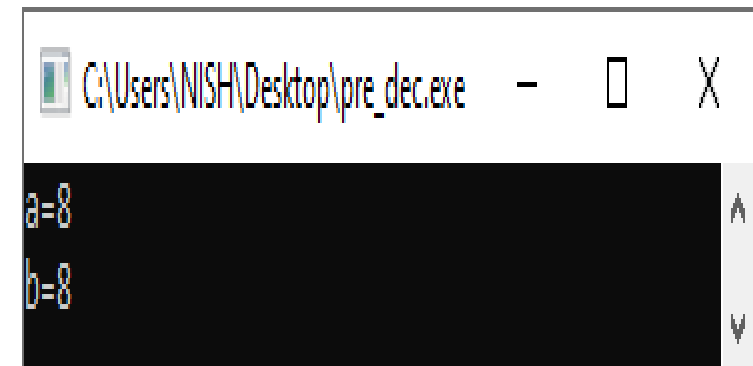
```
#include<stdio.h>
void main()
{
    int a=9,b;
    b=a--;
    printf("a=%d\n",a);
    printf("b=%d",b);
}
```



```
C:\Users\NISH\Desktop\post_dec.exe
a=8
b=9
```

Prefix Decrement

```
#include<stdio.h>
void main()
{
    int a=9,b;
    b=--a;
    printf("a=%d\n",a);
    printf("b=%d",b);
}
```



```
C:\Users\NISH\Desktop\pre_dec.exe
a=8
b=8
```

Increment & Decrement operators

```
#include<stdio.h>
void main()
{
    int x = 10, y = 20, a = 5, b = 4;

    printf("---- PRE INCREMENT OPERATOR EXAMPLE---- \n");
    printf("Value of x : %d \n", x); //Original Value
    printf("Value of x : %d \n", ++x); // using Pre increment Operator
    printf("Value of x Incremented : %d \n", x); //Incremented value

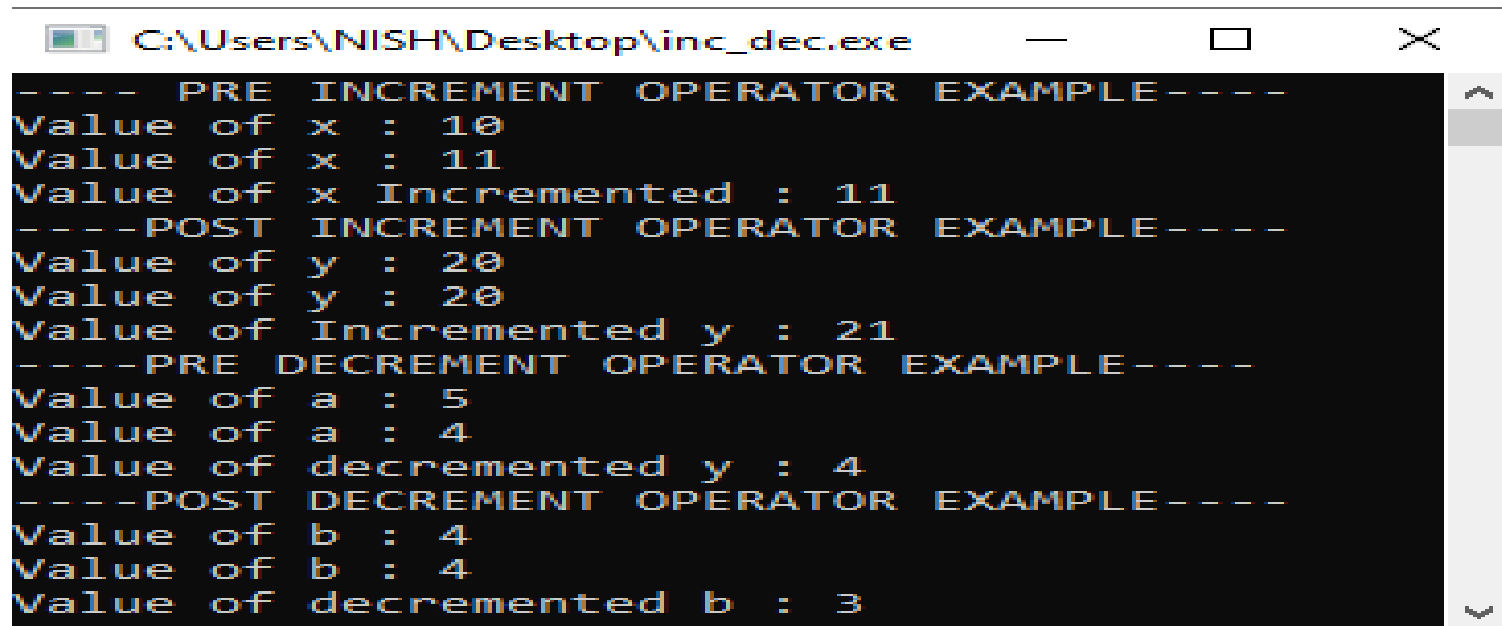
    printf("----POST INCREMENT OPERATOR EXAMPLE---- \n");
    printf("Value of y : %d \n", y); //Original Value
    printf("Value of y : %d \n", y++); // using Post increment Operator
    printf("Value of Incremented y : %d \n", y); //Incremented value

    printf("----PRE DECREMENT OPERATOR EXAMPLE---- \n");
    printf("Value of a : %d \n", a); //Original Value
    printf("Value of a : %d \n", --a); // using Pre decrement Operator
    printf("Value of decremented y : %d \n", a); //decremented value

    printf("----POST DECREMENT OPERATOR EXAMPLE---- \n");
    printf("Value of b : %d \n", b); //Original Value
    printf("Value of b : %d \n", b--); // using Post decrement Operator
    printf("Value of decremented b : %d \n", b); //decremented value
}
```

Increment & Decrement operators

Output



```
C:\Users\NISH\Desktop\inc_dec.exe
---- PRE INCREMENT OPERATOR EXAMPLE ----
Value of x : 10
Value of x : 11
Value of x Incremented : 11
----POST INCREMENT OPERATOR EXAMPLE----
Value of y : 20
Value of y : 20
Value of Incremented y : 21
----PRE DECREMENT OPERATOR EXAMPLE----
Value of a : 5
Value of a : 4
Value of decremented y : 4
----POST DECREMENT OPERATOR EXAMPLE----
Value of b : 4
Value of b : 4
Value of decremented b : 3
```

Increment & Decrement operators



Find the output of the following Program:

```
#include<stdio.h>

void main()
{
    int i=-3, j=2, k=0, m;

    m= ++i || ++j && ++k;

    printf("%d %d %d %d", i, j, k, m);
}
```

Conditional / ternary operators

A ternary operator pair “ **?:** ” is used to construct conditional expressions as shown below:

Syntax:

condition? True_value : False_value

Example:

(a>b)?a:b;

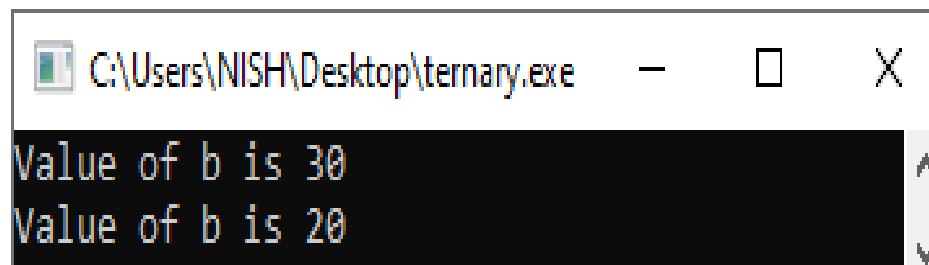
Conditional / ternary operators

```
#include<stdio.h>

void main()
{
    int a = 10,b;

    b = (a == 1) ? 20: 30;
    printf( "Value of b is %d\n", b );

    b = (a == 10) ? 20: 30;
    printf( "Value of b is %d\n", b );
}
```



Conditional / ternary operators

C Program to find the greatest of the two numbers using conditional operators.

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

```
void main()
```

```
{
```

```
    int a,b,c;
```

```
    printf("\n Enter First Number : ");
```

```
    scanf("%d",&a);
```

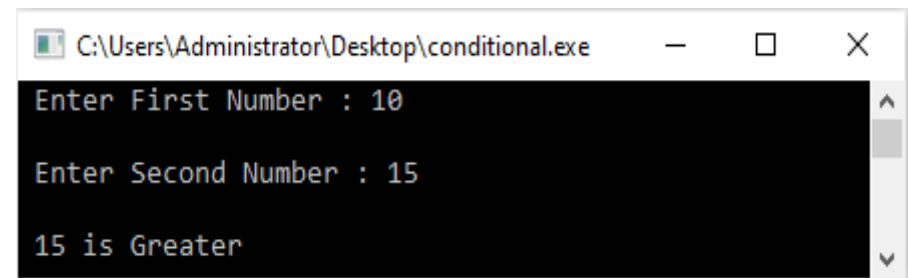
```
    printf("\n Enter Second Number : ");
```

```
    scanf("%d",&b);
```

```
    c=(a>b) ? a : b ;    // Conditional operator
```

```
    printf("\n %d is Greater",c);
```

```
}
```



```
C:\Users\Administrator\Desktop\conditional.exe
Enter First Number : 10
Enter Second Number : 15
15 is Greater
```

Conditional / ternary operators



Practical-3.1: Write a program to find the greatest of the three numbers entered through the keyboard using conditional operators.

Bit wise operators

used to perform bit operations.

Operator	Description
&	bitwise AND
	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	one's complement

x	y	x y	x&y	x^y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Bit wise operators

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

```
void main()
```

```
{
```

```
    int m = 40, n = 80, AND_opr, OR_opr, XOR_opr, NOT_opr ;
```

```
    AND_opr = (m&n);
```

```
    OR_opr = (m|n);
```

```
    NOT_opr = (~m);
```

```
    XOR_opr = (m^n);
```

```
    printf("AND_opr value = %d\n", AND_opr );
```

```
    printf("OR_opr value = %d\n", OR_opr );
```

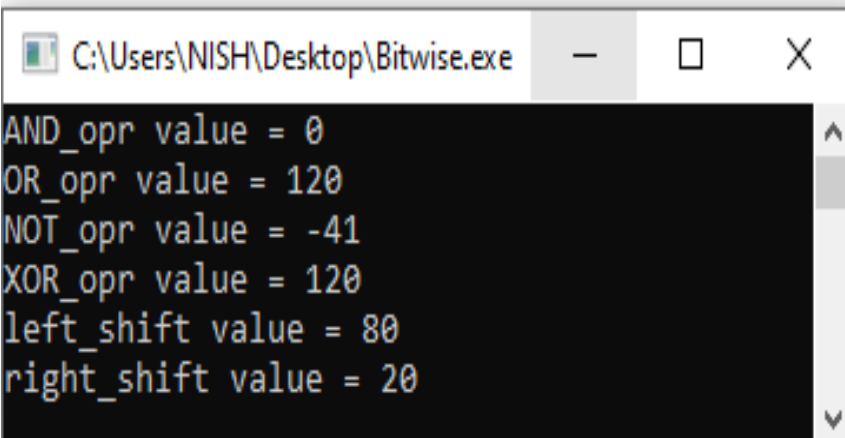
```
    printf("NOT_opr value = %d\n", NOT_opr );
```

```
    printf("XOR_opr value = %d\n", XOR_opr );
```

```
    printf("left_shift value = %d\n", m << 1);
```

```
    printf("right_shift value = %d\n", m >> 1);
```

```
}
```



```
C:\Users\NISH\Desktop\Bitwise.exe
AND_opr value = 0
OR_opr value = 120
NOT_opr value = -41
XOR_opr value = 120
left_shift value = 80
right_shift value = 20
```

Special operators

1. comma operator (,)
2. sizeof operator (sizeof())
3. Reference operator or Address Operator (&)
4. Dereference operator or Pointer Operator (*)
5. Double Pointer operator (**)
6. member selection operator (. and ->)

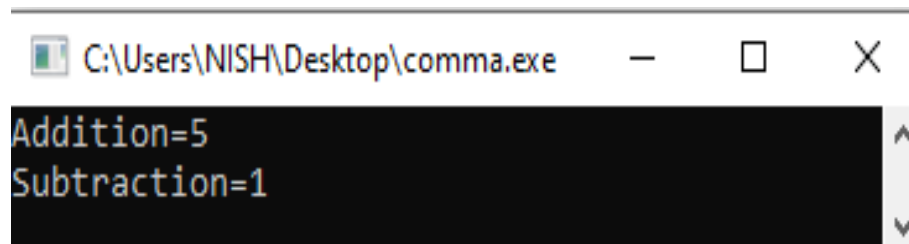
comma operator

Used to link related expressions together.

Example:

```
int a, b=5, c;
```

```
#include<stdio.h>
void main()
{
    printf("Addition=%d\nSubtraction=%d", 2+3, 5-4);
}
```

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\NISH\Desktop\comma.exe" and standard window controls (minimize, maximize, close). The command prompt has a black background and displays the output of the program: "Addition=5" on the first line and "Subtraction=1" on the second line.

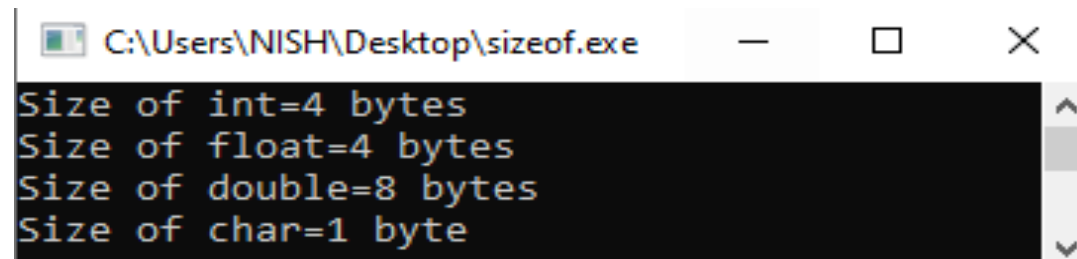
```
C:\Users\NISH\Desktop\comma.exe
Addition=5
Subtraction=1
```

sizeof operator

Returns the size of data(constants, variables, array, structure, etc)

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%d bytes\n", sizeof(a));
    printf("Size of float=%d bytes\n", sizeof(b));
    printf("Size of double=%d bytes\n", sizeof(c));
    printf("Size of char=%d byte\n", sizeof(d));

    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\NISH\Desktop\sizeof.exe". The window has standard Windows window controls (minimize, maximize, close). The output of the program is displayed in the command prompt:

```
Size of int=4 bytes
Size of float=4 bytes
Size of double=8 bytes
Size of char=1 byte
```

Operators

Based on number of operands, operators are categorized as follows:

1. Unary Operators
2. Binary Operators
3. Ternary Operators

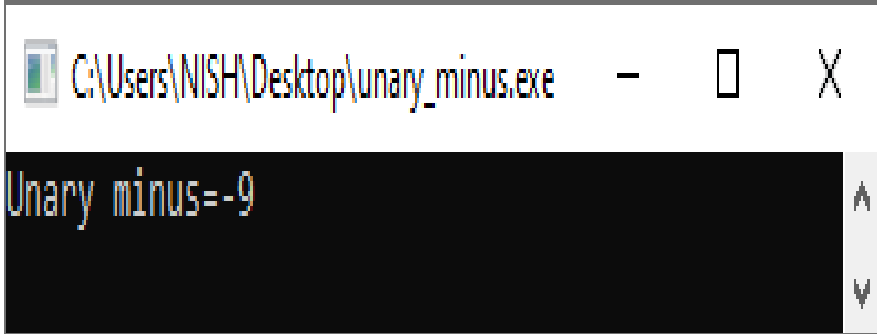
Unary Operators

Operators which require **single operand** to perform any action

1. unary minus(-)
2. unary plus(+)
3. increment(++)
 - post-increment
 - pre-increment
4. decrement(--)
 - post-decrement
 - pre-decrement
5. Logical NOT(!)
6. Bitwise NOT(~)
7. Address-of(&)
8. Pointer dereference(*)
9. cast operator()
10. sizeof()

unary-minus

```
#include<stdio.h>
void main()
{
    int a=9;
    printf("Unary minus=%d\n",-a);
}
```



Binary Operators

Operators which require **two operand** to perform any action

1. Arithmetic Operators (+ , - , * , / , %)
2. Logical Operators (&& , ||)
3. Relational Operators (== , != , > , < , >= , <=)
4. Bitwise Operators (& , | , ^ , >> , <<)
5. Assignment Operators(=, += , -= , *= , /= , %= , &= , ^= , |=)
6. Special Operators
 1. Comma (,)
 2. Member selection(->)
 3. Pointer to member selection(->*)

Ternary Operators

Operators which require **three operand** to perform any action

- A ternary operator is a short form of if-else.

Practical-3.1: Write a program to find the greatest of the three numbers entered through the keyboard using conditional operators.

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

```
void main()
```

```
{
```

```
    int a, b, c, big ;
```

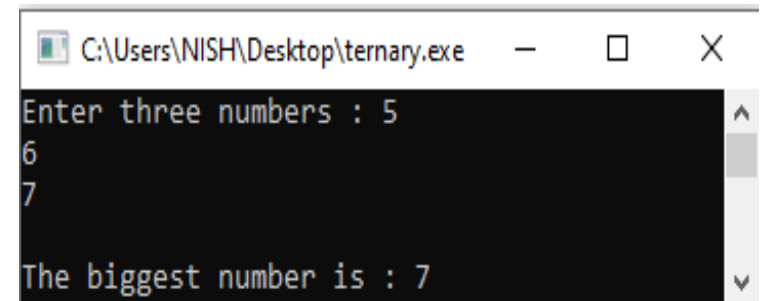
```
    printf("Enter three numbers : ") ;
```

```
    scanf("%d %d %d", &a, &b, &c) ;
```

```
    big = a > b ? (a > c ? a : c) : (b > c ? b : c) ;
```

```
    printf("\nThe biggest number is : %d", big) ;
```

```
}
```



```
C:\Users\NISH\Desktop\ternary.exe
Enter three numbers : 5
6
7
The biggest number is : 7
```

Arithmetic Expressions

An expression is a **combination** of **variables**, **constants**, and **operators** arranged as per the syntax of the language.

Note: C does not have an operator for exponentiation.

Algebraic Expression	C Expression
$a \times b - c \times d$	$a * b - c * d$
$(m + n) (a + b)$	$(m + n) * (a + b)$
$3x^2 + 2x + 5$	$3 * x * x + 2 * x + 5$
$\frac{a + b + c}{d + e}$	$(a + b + c) / (d + e)$
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)} \right]$	$2 * b * y / (d + 1) - x / 3 * (z + y)$

Evaluation of Expressions

Expressions are evaluated using an **assignment** statement

Syntax:

variable = expression;

Example:

x = a * b - c;

y = b / c * a;

z = a - b / c + d;

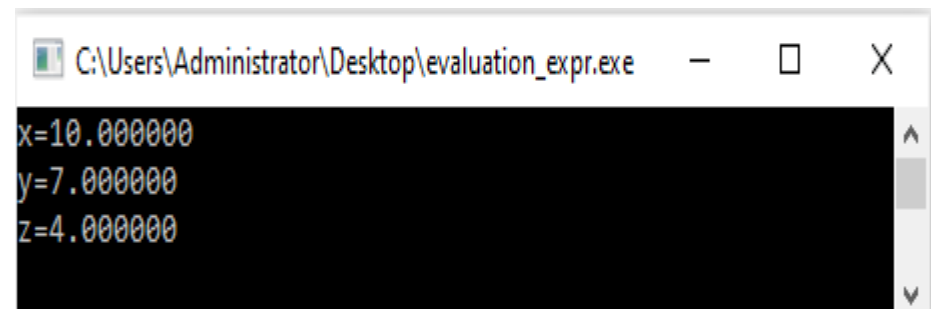
Evaluation of Expressions

```
#include<stdio.h>

void main()
{
    float a,b,c,x,y,z;
    a=9;
    b=12;
    c=3;

    x=a-b/3+c*2-1;
    y=a-b/(3+c)*(2-1);
    z=a-(b/(3+c)*2)-1;

    printf("x=%f\n",x);
    printf("y=%f\n",y);
    printf("z=%f\n",z);
}
```



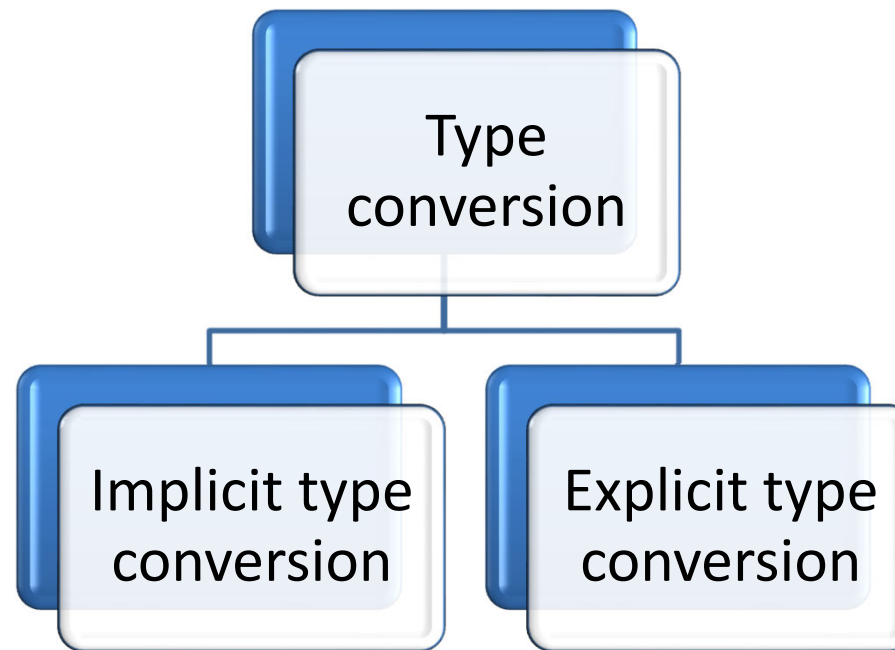
A screenshot of a Windows command prompt window titled "C:\Users\Administrator\Desktop\evaluation_expr.exe". The window displays the output of the C program: x=10.000000, y=7.000000, and z=4.000000, each on a new line.

```
C:\Users\Administrator\Desktop\evaluation_expr.exe
x=10.000000
y=7.000000
z=4.000000
```

Type Conversion

- Converting the **data type of one operand to another** is called as **type conversion**.
- An expression has different types of operands, which can be same or different data types.
- If they are of same data type no type conversion is required.
- But if they are of different type, then they must be converted to a single data type of destination data type.

Types of Type Conversion



Implicit Type Conversion

- Implicit type conversion is done **automatically by the compiler** itself without an user intervention.
- The data type of all the operands are **automatically converted** in to the data type of the operand with the **largest data type**.

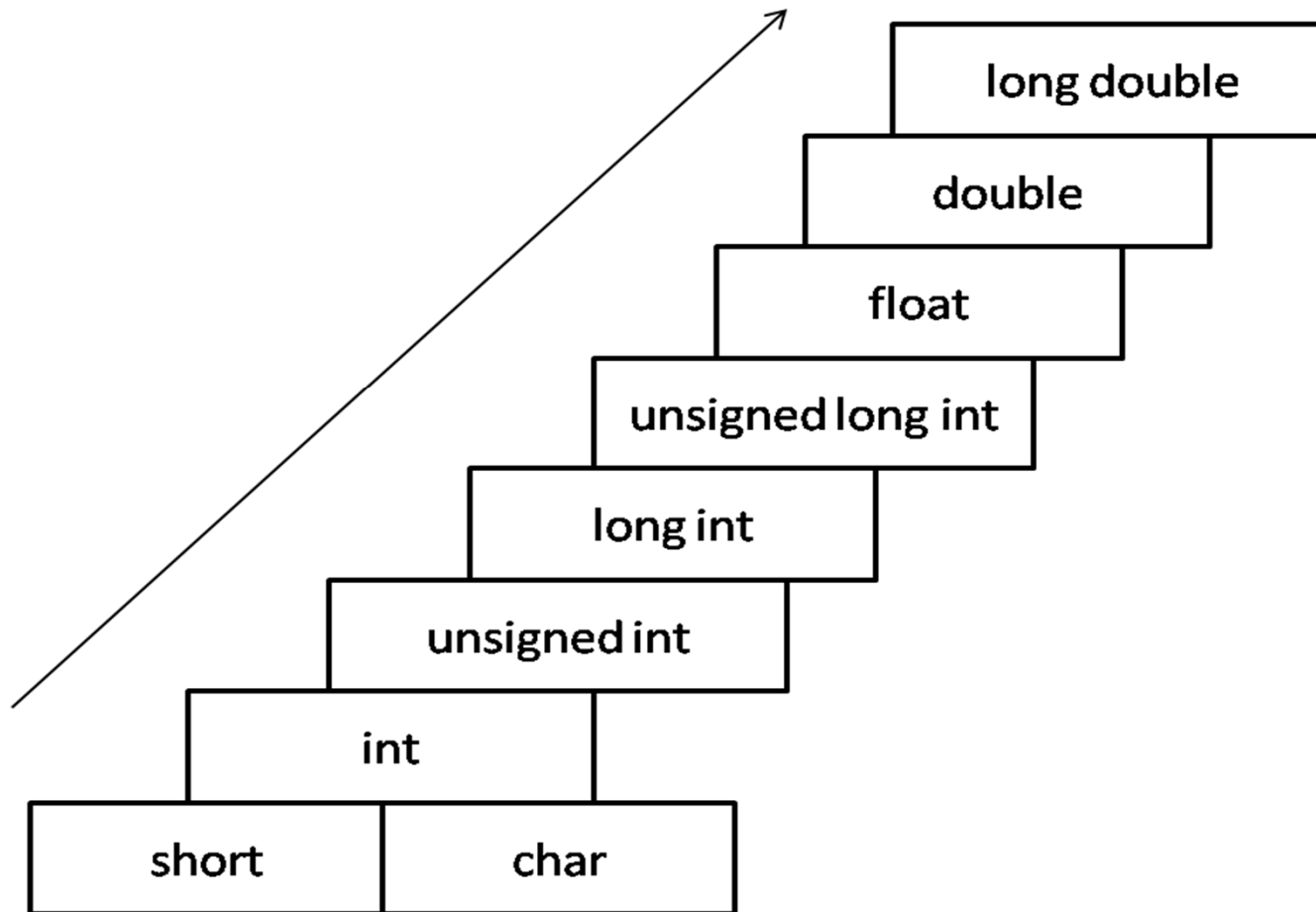
Example:

```
int a=5;  
float b=2.5,c;  
c=a + b;
```

*/*here, float is the largest data type, so a is implicitly converted to type float*/*

Implicit Type Conversion

Conversion Hierarchy



Implicit Type Conversion

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

```
void main()  
{
```

```
    int x;
```

```
    char y='A';
```

```
    float z;
```

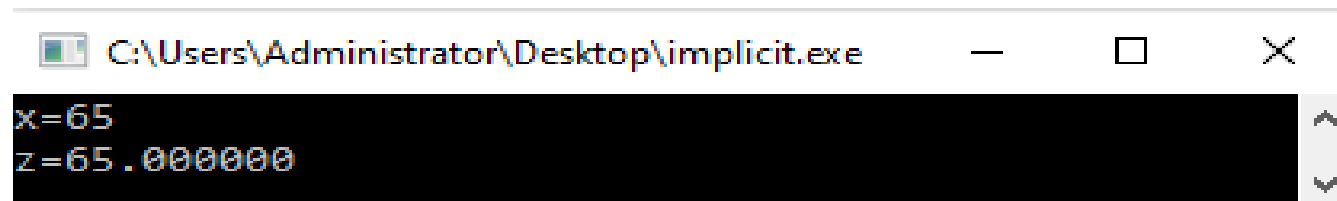
```
    x=y; //y value is automatically converted into int and the ASCII value of A is stored in x
```

```
    printf("x=%d",x);
```

```
    z=x; //x value is automatically converted into float and stored in z
```

```
    printf("\nz=%f",z);
```

```
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\Administrator\Desktop\implicit.exe". The window has standard Windows window controls (minimize, maximize, close). The output of the program is displayed in the command prompt:

```
x=65  
z=65.000000
```


Implicit Type Conversion

```
#include<stdio.h>

void main()
{
    int x=5;
    float y=2.0,z;

    z=x/y; //x is implicitly converted to float i.e z=5.0/2.0

    printf("z=%f",z);
}
```



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\Administrator\Desktop\implicit.exe" and standard window controls (minimize, maximize, close). The command prompt displays the output "z=2.500000".

Explicit Type Conversion

- In implicit type conversion, there is a **possibility of loss of data.**
- For example, when signed is converted in to unsigned, signs are lost.
- To overcome this drawback, explicit type conversion is used
- Explicit type conversion is done **explicitly by the user** by prefixing the operand or an expression with the data type to be converted
- It is also called as **type casting.**

Explicit Type Conversion

Syntax:

(data type) expression or operand;

Example	Action
<code>x=(int) 7.5</code>	7.5 is converted to integer by truncation
<code>a=(int)21.3/(int)4.5</code>	Evaluated as 21/4 and the result would be 5
<code>b=(double)sum/n</code>	Division is done in floating point mode
<code>y=(int) (a + b)</code>	The result of a + b is converted to integer
<code>z=(int) a + b</code>	a is converted to integer and then added to b
<code>p=cos((double)x)</code>	Converts x to double before using it

Explicit Type Conversion

```
#include<stdio.h>

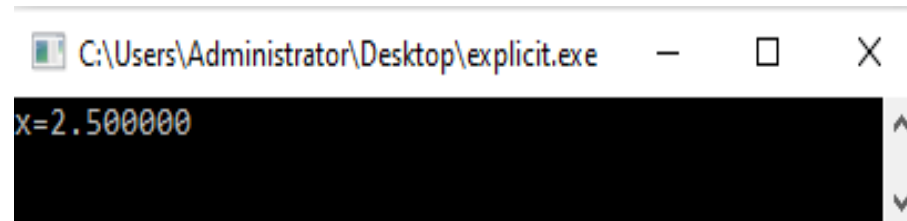
void main()
{
    int a,b;
    float x;

    a=10;
    b=4;

    //x=a/b;   Output will be 2.000000, loss of 0.5

    x=a/(float)b; /*Output will be 2.500000,no loss of data.
                  Also, b is explicitly converted into float
                  and a will be automatically/implicitly converted into float*/

    printf("x=%f\n",x);
}
```



Type Conversion



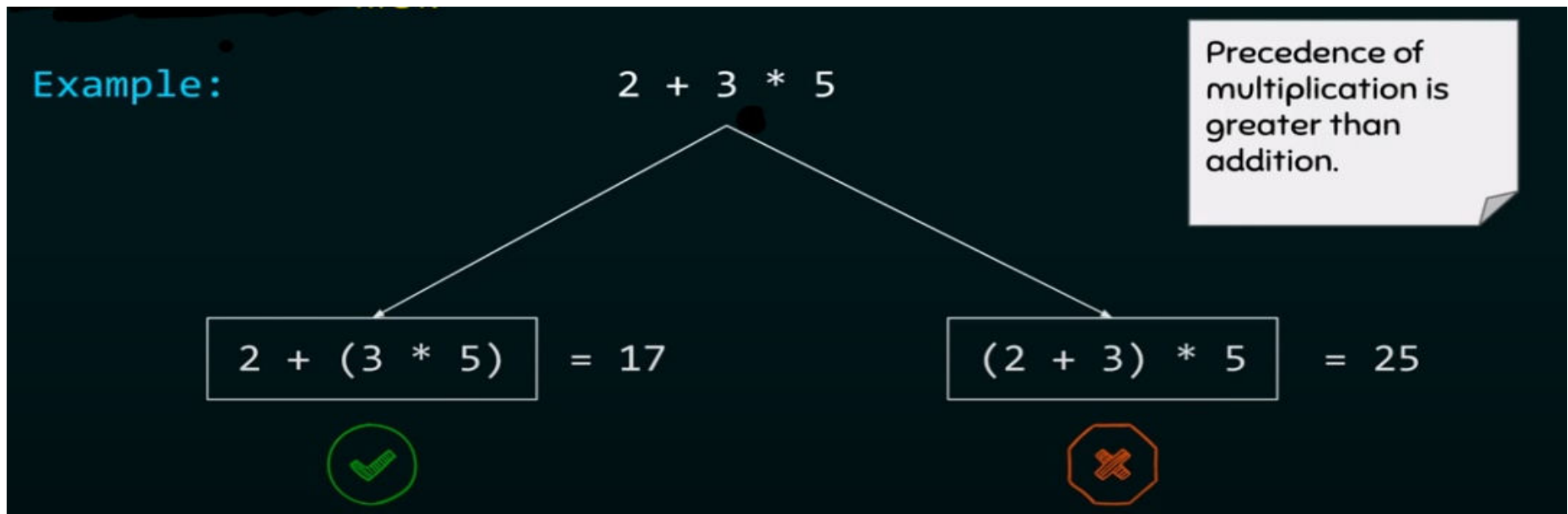
Write a program to do following:

- (a) Input an amount and convert it into rupees and paisa. (For Ex. 25.67 Rs = 25 Rs and 67 Paisa). (**Implicit type Conversion**)
- (b) Input No of female and No of male and calculate the ratio of females to males in a town. No of female and No of male are in int and ratio is in float. (For Ex. No_of_Female = 10 & No_of_Male = 7 then ratio = 1.43). (**Explicit type Conversion**)

Precedence and Associativity

Precedence of operators

It determines which operator is executed first if there is more than one operator in an expression.



Precedence and Associativity

Associativity of operators

It defines the order in which operators of the same precedence are evaluated in an expression

Example:

$10 / 2 * 5$

Left to right:

$(10 / 2) * 5 = 25$



Right to left:

$10 / (2 * 5) = 1$



Associativity can be either:

1. Left to right
- OR
2. Right to left

Precedence and Associativity

Operator	Description	Associativity
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to or not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
	Logical OR	left to right
?:	Ternary operator	right to left
= += -= *= /= %= &= ^= = <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	comma operator	left to right

Precedence and Associativity

```
#include<stdio.h>

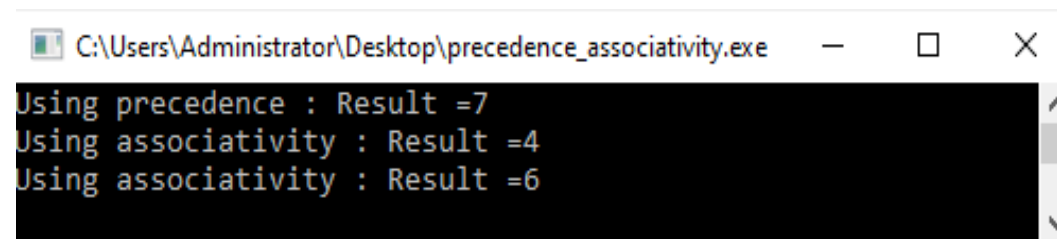
void main()
{
    int result;

    result= 5 + 4 / 2;
    printf("Using precedence : Result =%d\n",result);    //7

    result= 5 + 2 - 3;
    printf("Using associativity : Result =%d\n",result);    //4

    result= 5 - 2 + 3;
    printf("Using associativity : Result =%d\n",result);    //6

}
```

A screenshot of a Windows command prompt window titled "C:\Users\Administrator\Desktop\precedence_associativity.exe". The window has standard Windows window controls (minimize, maximize, close). The output of the program is displayed in three lines: "Using precedence : Result =7", "Using associativity : Result =4", and "Using associativity : Result =6". The text is white on a black background.

```
C:\Users\Administrator\Desktop\precedence_associativity.exe
Using precedence : Result =7
Using associativity : Result =4
Using associativity : Result =6
```

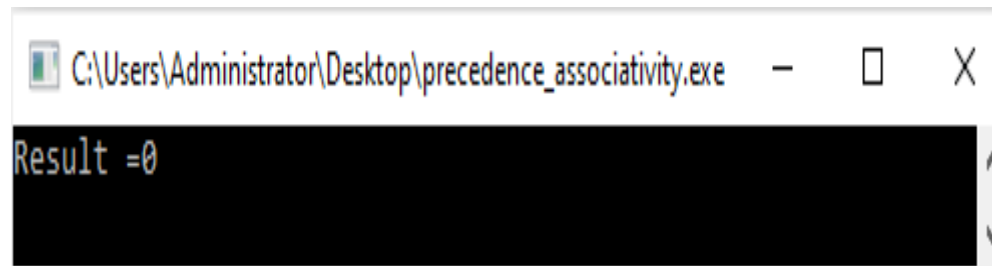
Precedence and Associativity

```
#include<stdio.h>

void main()
{
    int result;

    result= (6 + 8) / 7 > 10 && (7 + 7) < 10;
    //14 / 7 > 10 && 14 < 10
    //2 > 10 && 14 <10
    //0 && 0
    //0

    printf("Result =%d\n", result);
}
```

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\Administrator\Desktop\precedence_associativity.exe" and standard window controls (minimize, maximize, close). The command prompt area has a black background and displays the text "Result =0" in white.

Precedence and Associativity

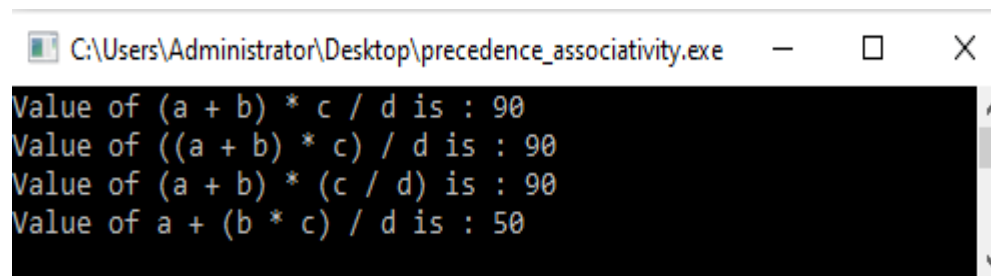
```
#include <stdio.h>
void main()
{
    int a = 20;
    int b = 10;
    int c = 15;
    int d = 5;
    int e;

    e = (a + b) * c / d; // ( 30 * 15 ) / 5
    printf("Value of (a + b) * c / d is : %d\n", e );

    e = ((a + b) * c) / d; // (30 * 15 ) / 5
    printf("Value of ((a + b) * c) / d is : %d\n" , e );

    e = (a + b) * (c / d); // (30) * (15/5)
    printf("Value of (a + b) * (c / d) is : %d\n", e );

    e = a + (b * c) / d; // 20 + (150/5)
    printf("Value of a + (b * c) / d is : %d\n" , e );
}
```



```
C:\Users\Administrator\Desktop\precedence_associativity.exe
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50
```

Mathematical Functions

- Mathematical functions such as **cos, sqrt , log** , etc. are frequently used in analysis of real-life problems
- To use any of math functions in a program, one should include below header file in the beginning of the program

#include <math.h>

- All the functions available in this library take double as an argument and return double as the result.

Mathematical Functions

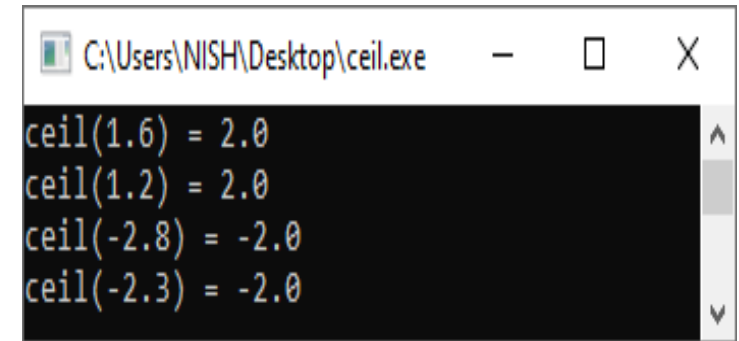
Function	Meaning
Trigonometric acos(x) asin(x) atan(x) atan2(x,y) cos(x) sin(x) tan(x)	Arc cosine of x Arc sine of x Arc tangent of x Arc tangent of x/y cosine of x sine of x tangent of x
Hyperbolic cosh(x) sinh(x) tanh(x)	Hyperbolic cosine of x Hyperbolic sine of x Hyperbolic tangent of x
Other functions ceil(x) exp(x) fabs(x) floor(x) fmod(x,y) log(x) log10(x) pow(x,y) sqrt(x)	x rounded up to the nearest integer e to the power x absolute value of x x rounded down to the nearest integer remainder of x/y natural log of x, $x > 0$ base 10 log of x, $x > 0$ x to the power y square root of x, $x \geq 0$

Mathematical Functions

ceil(x)

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

void main()
{
    printf ("ceil(1.6) = %.11f\n", ceil(1.6));
    printf ("ceil(1.2) = %.11f\n", ceil(1.2));
    printf ("ceil(-2.8) = %.11f\n", ceil(-2.8));
    printf ("ceil(-2.3) = %.11f\n", ceil(-2.3));
}
```

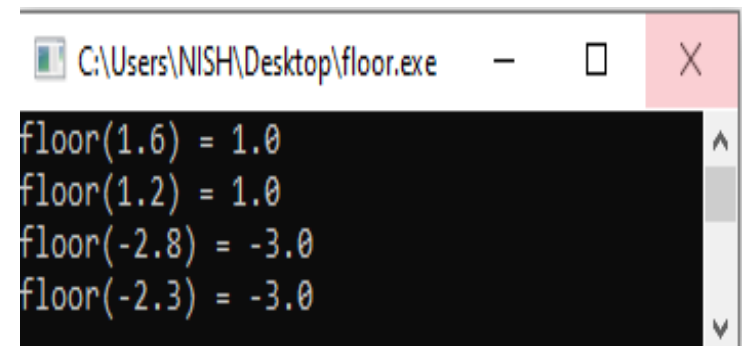


```
C:\Users\NISH\Desktop\ceil.exe
ceil(1.6) = 2.0
ceil(1.2) = 2.0
ceil(-2.8) = -2.0
ceil(-2.3) = -2.0
```

floor(x)

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

void main()
{
    printf ("floor(1.6) = %.11f\n", floor(1.6));
    printf ("floor(1.2) = %.11f\n", floor(1.2));
    printf ("floor(-2.8) = %.11f\n", floor(-2.8));
    printf ("floor(-2.3) = %.11f\n", floor(-2.3));
}
```



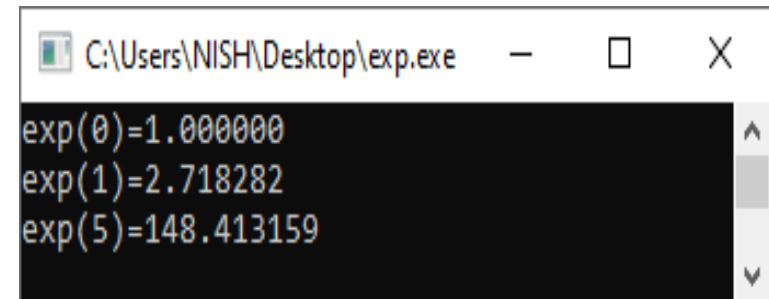
```
C:\Users\NISH\Desktop\floor.exe
floor(1.6) = 1.0
floor(1.2) = 1.0
floor(-2.8) = -3.0
floor(-2.3) = -3.0
```

Mathematical Functions

exp(x)

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

void main ()
{
    printf("exp(0)=%lf\n", exp(0));
    printf("exp(1)=%lf\n", exp(1));
    printf("exp(5)=%lf\n", exp(5));
}
```

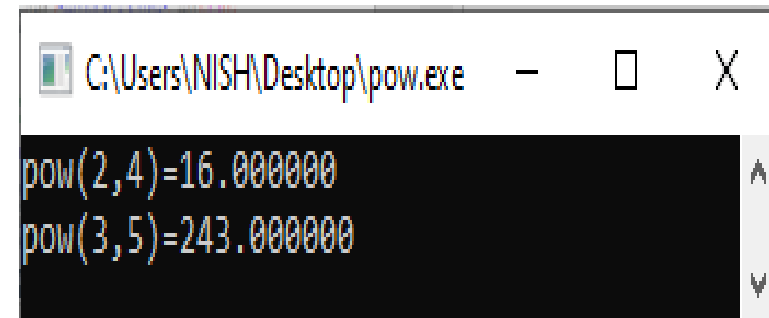


exp(0)=1.000000
exp(1)=2.718282
exp(5)=148.413159

pow(x,y)

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

void main ()
{
    printf("pow(2,4)=%lf\n", pow(2,4));
    printf("pow(3,5)=%lf\n", pow(3,5));
}
```



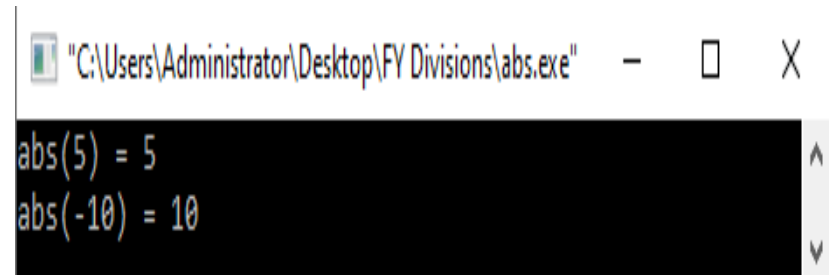
pow(2,4)=16.000000
pow(3,5)=243.000000

Mathematical Functions

abs(x)

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

void main()
{
    printf("abs(5) = %d\n", abs(5));
    printf("abs(-10) = %d\n", abs(-10));
}
```



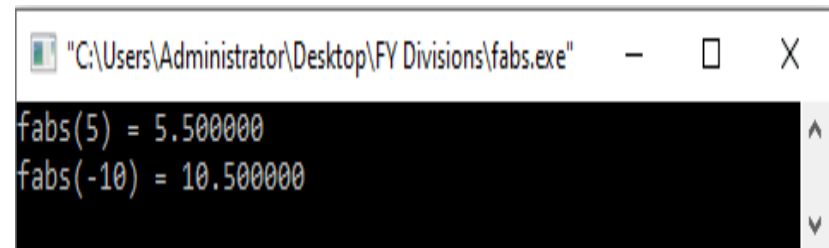
The screenshot shows a command prompt window titled "C:\Users\Administrator\Desktop\FY Divisions\abs.exe". The output displayed is:

```
abs(5) = 5
abs(-10) = 10
```

fabs(x)

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

void main()
{
    printf("fabs(5) = %lf\n", fabs(5.5));
    printf("fabs(-10) = %lf\n", fabs(-10.5));
}
```



The screenshot shows a command prompt window titled "C:\Users\Administrator\Desktop\FY Divisions\fabs.exe". The output displayed is:

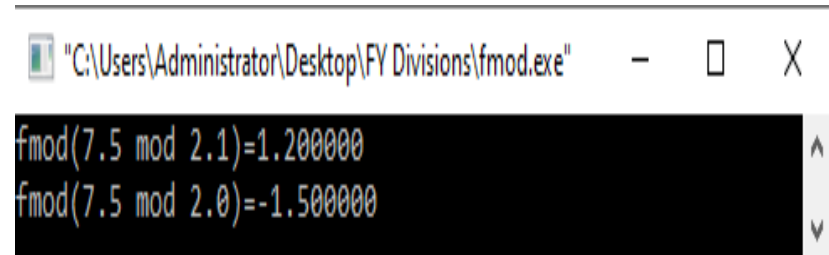
```
fabs(5) = 5.500000
fabs(-10) = 10.500000
```

Mathematical Functions

fmod(x,y)

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

```
void main()
{
    printf("fmod(7.5 mod 2.1)=%lf\n", fmod(7.5, 2.1));
    printf("fmod(7.5 mod 2.0)=%lf", fmod(-17.50, 2.0));
}
```

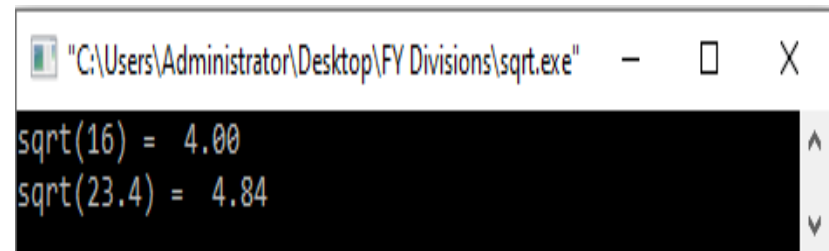


```
"C:\Users\Administrator\Desktop\FY Divisions\fmod.exe"
fmod(7.5 mod 2.1)=1.200000
fmod(7.5 mod 2.0)=-1.500000
```

sqrt(x)

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

```
void main()
{
    printf("sqrt(16) = %.2lf\n", sqrt(16));
    printf("sqrt(23.4) = %.2lf", sqrt(23.4));
}
```



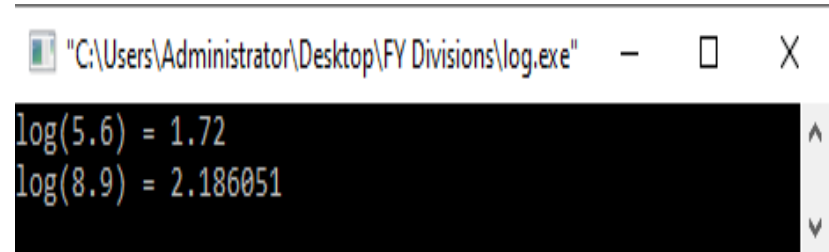
```
"C:\Users\Administrator\Desktop\FY Divisions\sqrt.exe"
sqrt(16) = 4.00
sqrt(23.4) = 4.84
```

Mathematical Functions

log(x)

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

```
void main()
{
    printf("log(5.6) = %.2lf\n", log(5.6));
    printf("log(8.9) = %lf\n", log(8.9));
}
```

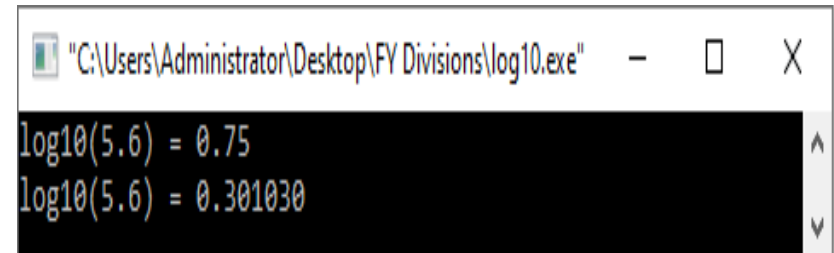


```
"C:\Users\Administrator\Desktop\FY Divisions\log.exe"
log(5.6) = 1.72
log(8.9) = 2.186051
```

log10(x)

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

```
void main()
{
    printf("log10(5.6) = %.2f\n", log10(5.6));
    printf("log10(5.6) = %lf\n", log10(2));
}
```



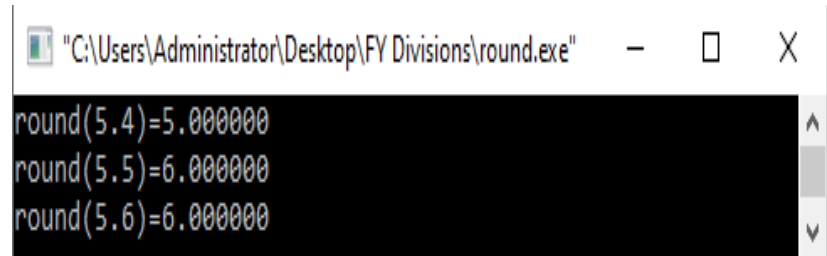
```
"C:\Users\Administrator\Desktop\FY Divisions\log10.exe"
log10(5.6) = 0.75
log10(5.6) = 0.301030
```

Mathematical Functions

round(x)

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

```
void main()
{
    printf("round(5.4)=%lf\n", round(5.4));
    printf("round(5.5)=%lf\n", round(5.5));
    printf("round(5.6)=%lf\n", round(5.6));
}
```

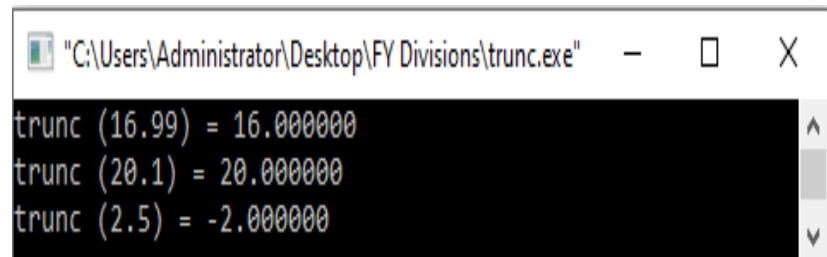


```
"C:\Users\Administrator\Desktop\FY Divisions\round.exe"
round(5.4)=5.000000
round(5.5)=6.000000
round(5.6)=6.000000
```

trunc(x)

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

```
void main()
{
    printf("trunc (16.99) = %lf\n", trunc (16.99) );
    printf("trunc (20.1) = %lf\n", trunc (20.1) );
    printf("trunc (2.5) = %lf\n", trunc (-2.5) );
}
```



```
"C:\Users\Administrator\Desktop\FY Divisions\trunc.exe"
trunc (16.99) = 16.000000
trunc (20.1) = 20.000000
trunc (2.5) = -2.000000
```

End of Unit-03

