BCSE102L- Structured and object-oriented programming

Module 1

Dr. P.Keerthika
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

BCSE102P- Structured and object-oriented programming Laboratory

Indicative Experiments	
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

BCSE102L- Structured and Object-Oriented Programming

- **Module-1:C Program Fundamentals - Introduction**
  - **Variables**
  - **Reserved Words**
  - **Data types**
  - **Operators- Operator Precedence**
  - **Expressions**
  - **Type Conversions**
  - **I/O Statements**

BCSE102L- Structured and Object-Oriented Programming

- **Module-1: C Program Fundamentals – Branching and Looping**
  - if, if-else, nested if, else-if ladder
  - Switch Statement
  - Goto Statement
  - Looping
    - For
    - While and do while
  - Break Statement
  - Continue Statement

OPERATORS AND EXPRESSIONS

- **C supports a rich set of built-in operators**
  - For Example: Basically we used =,+,-,*,/, & ……. etc
  - **Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.**
  - **Arithmetic**
  - **Relational**
  - **Logical**
  - **Assignment**
  - **Bitwise**

  → ***Binary Operators***

  - **Increment and Decrement** → ***Unary Operator***
  - **Conditional** → ***Ternary Operator***
  - **Special**

Arithmetic Operators

- **Perform basic arithmetic operations**

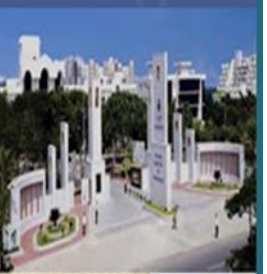
Operator	Name	Example
+	Addition	12 + 4.9 // gives 16.9
-	Subtraction	3.98 - 4 // gives -0.02
*	Multiplication	2 * 3.4 // gives 6.8
/	Division	9 / 2.0 // gives 4.5
%	Remainder	13 % 3 // gives 1

- **Short hand Operators (++, --)**

  ++ ➔ increment by 1, -- ➔ Decrement by 1

  e.g. a++ is equivalent to a = a + 1
  b-- is equivalent to b = b – 1

Note: % operator cannot be used with real operands

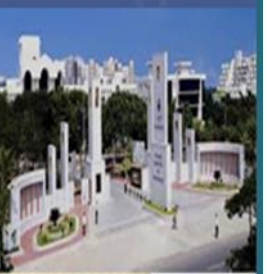


Relational Operators

- **Compare two quantities and depending on their relation, taking certain decisions.**

Operator	Name	Example
==	Equality	5 == 5 // gives 1
!=	Inequality	5 != 5 // gives 0
<	Less Than	5 < 5.5 // gives 1
<=	Less Than or Equal	5 <= 5 // gives 1
>	Greater Than	5 > 5.5 // gives 0
>=	Greater Than or Equal	6.3 >= 5 // gives 1

- **It is used in decision making statements to decide the course of action of a running program.**



Logical Operators

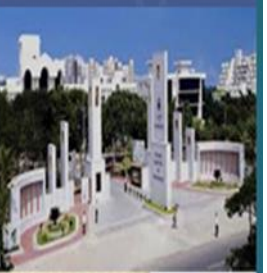
- Join multiple conditions or negate a condition i.e., Expression of this kind which combines two or more relational expressions.

Operator	Name	Example	
!	Logical Negation	!(5 == 5)	// gives 0
&&	Logical And	5 < 6 && 6 < 6	// gives 1
||	Logical Or	5 < 6 || 6 < 5	// gives 1

Truth Table for ! (not) Operator

Expression	!(Expression)
true	false
false	true

True in C means non-zero & False means zero



Logical Operators

Truth Table for AND

Expression1	Expression2	Expression1 && Expression2
true	true	true
true	false	false
false	true	false
false	false	false

Truth Table for OR

Expression1	Expression2	Expression1 || Expression2
true	true	true
true	false	true
false	true	true
false	false	false

Logical Operators

- **Short Circuit operations:**
  - <condition-1> && <condition-2>
    - If condition 1 is false, then condition 2 will not be checked at all
  - <condition 1> || <condition 2>
    - If condition 1 is true, then condition 2 will not be checked at all

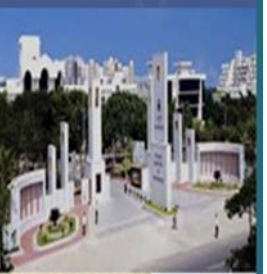
For Example:

i. To check For The Largest among Three Numbers

(First_no > Second_no) && (First_no > Third_no)

ii. To Check for the value of choice is either 'y' or 'Y'

(Choice == 'y') || (Choice == 'Y')



Assignment Operators

- "=" **used to assign the result of an expression to a variable**

  For Example:

  Number = 100;

  Temp = Number;

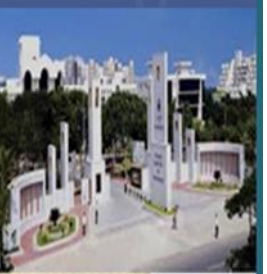
- **Shorthand Operators (op=)**
  - Op may be any arithmetic operator. For Example:
  - **Assignment += 10;** This statement is equivalent to **Assignment = Assignment + 10;**

Statement with simple assignment Operator	Statement with short hand Operator
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

Bitwise Operators

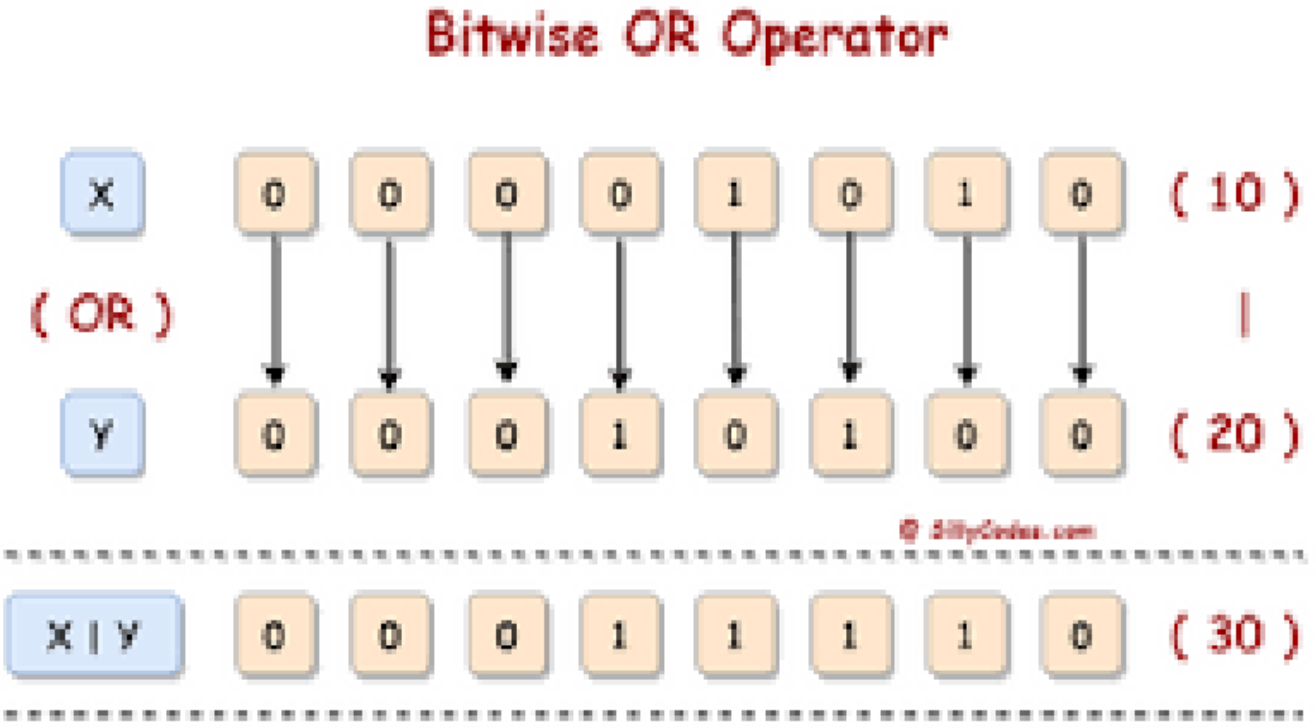
- Used for manipulation of data's at bit level.
- It is mainly used in numerical computations for a faster calculation because it consists of two digits – 1 or 0.
- Used for testing the bits, or shifting them right or left.
  - **Note: Bitwise operators not applied to float or double**

Operation	Meaning
x & y	Bitwise AND
x | y	Bitwise OR
x ^ y	Bitwise XOR
~x	Invert all bits of x
x >> y	Shift all bits of x y positions to the right
x << y	Shift all bits of x y positions to the left



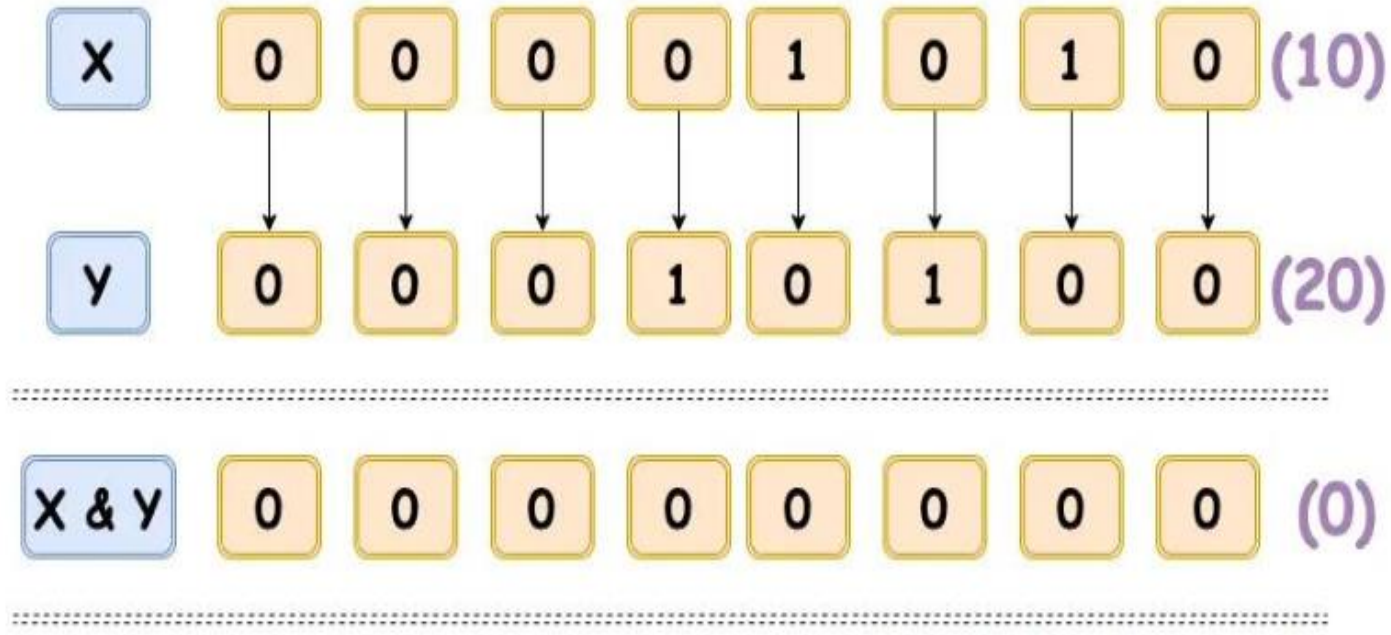
Bitwise Operators(OR)

- Bitwise Inclusive OR (|)
  - Bit by bit OR ing is done using the truth table of OR



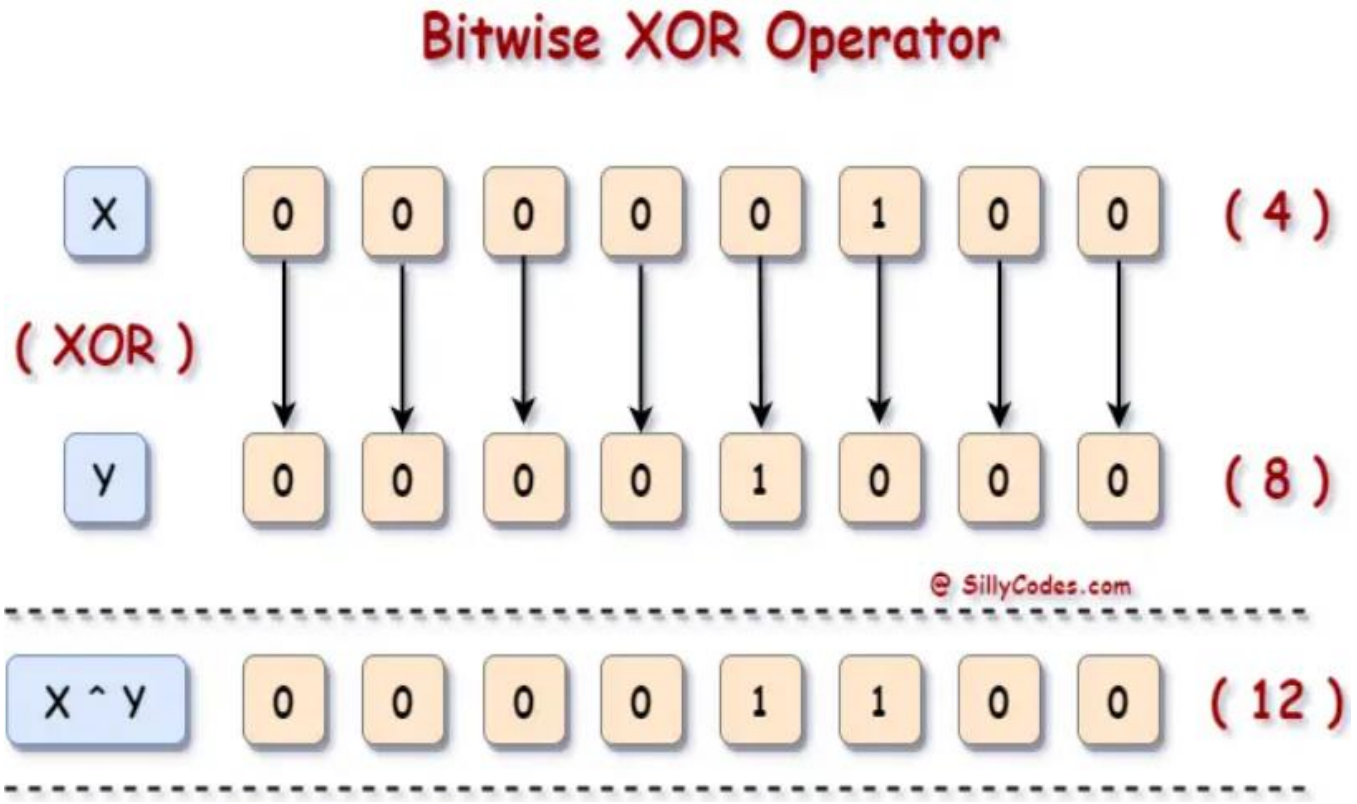
Bitwise Operators(AND)

- Bitwise AND(&)
  - Bit by bit AND ing is done using the truth table of AND



Bitwise Operators(XOR)

- Bitwise XOR(^)
  - Bit by bit XOR ing is done using the truth table of XOR

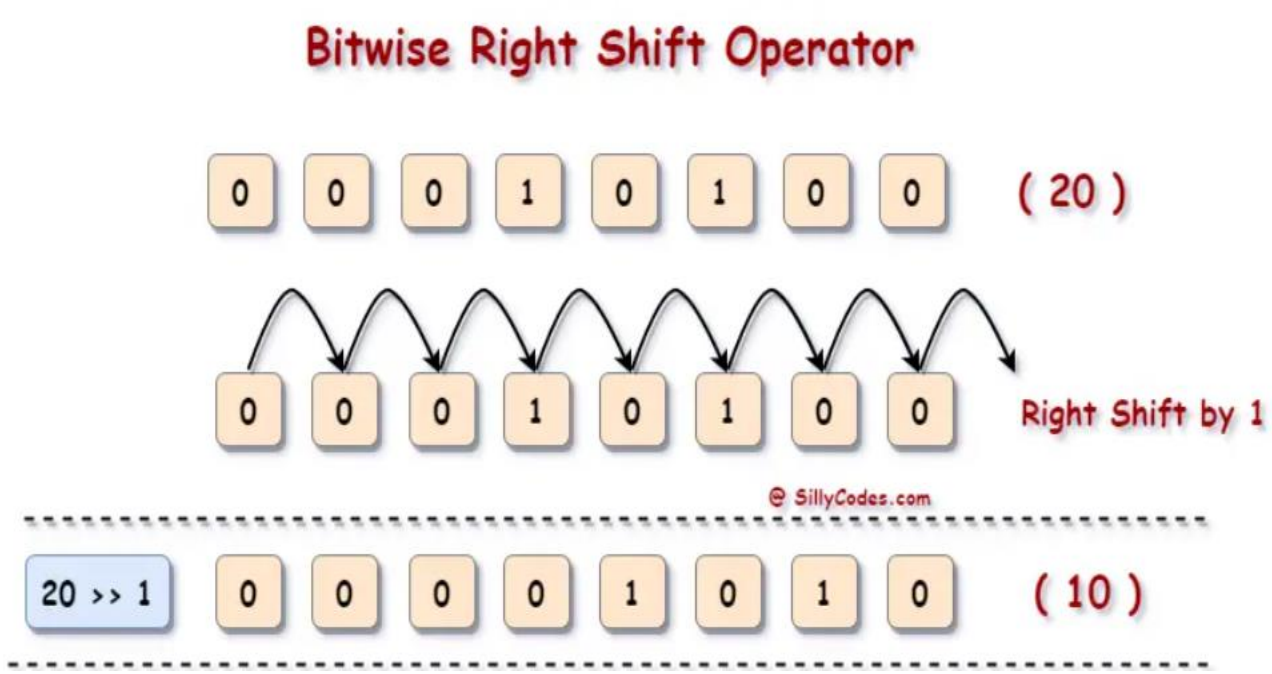


Bitwise One's Complement Operator(~)

- Bitwise **One's Complement Operator(~)**
  - Bit by bit converts all the zero(0) bits to One(1) and All One(1) bits to Zero(0).

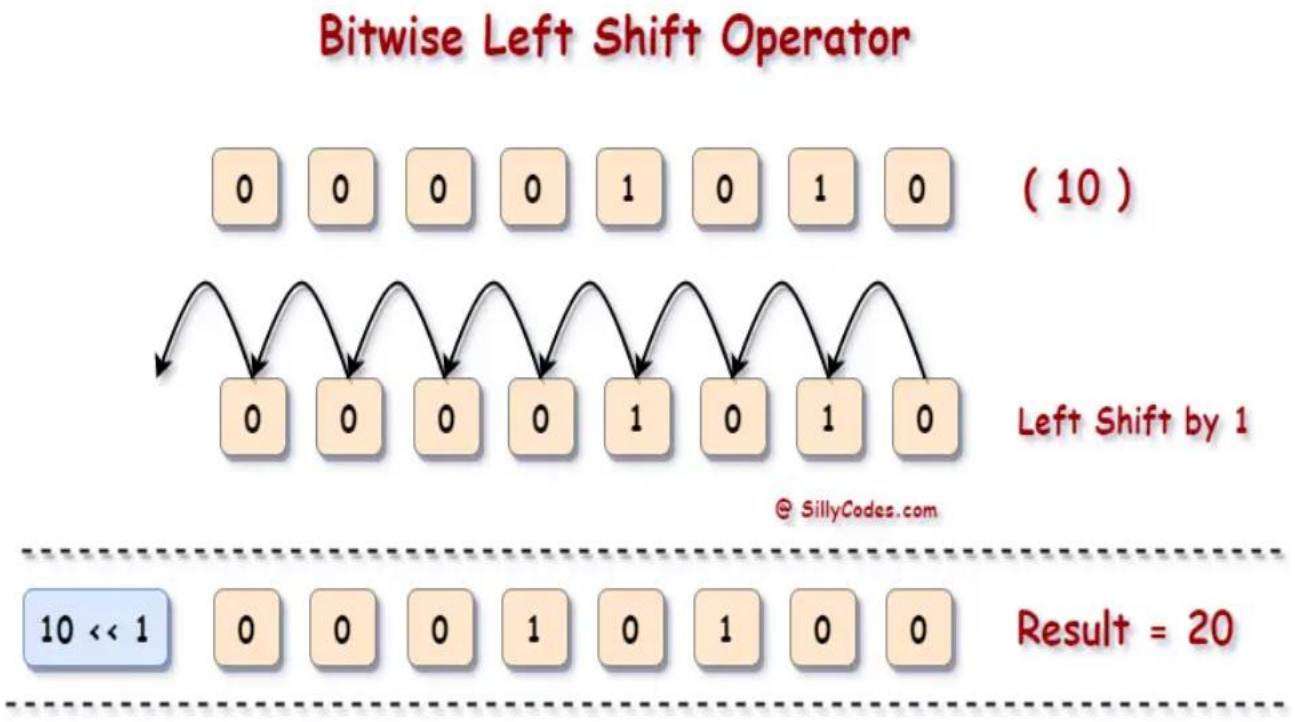
Bitwise Right Shift Operator(>>)

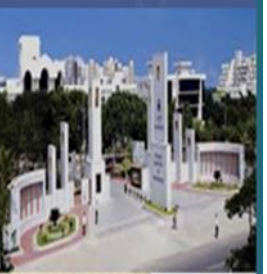
- Move(shift) the specific number of bits in binary sequence in the right direction.
- Syntax: **number >> number_of_positions_to_shift**



Bitwise Left Shift Operator(<<)

- Move(shift) the specific number of bits in binary sequence in the left direction.
- Syntax: **number << number_of_positions_to_shift**





Increment and Decrement Operator(++ & --)

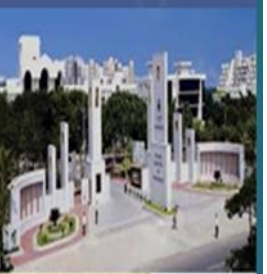
- Increment Operator(++) → ++x , x++
- **Increments the Value of the variable by 1.**
- **Two Categories**
  - **pre-increment operator** (or) *prefix increment operator.*
  - *Operator is before operand – increments value of the variable first, incremented values is taken for evaluation*

    X=10;

    Y=++X;
  - **post-increment operator** (or) *postfix increment operator.*
  - *Operator is after operand – first expression is evaluated and then the value of variable is incremented.*

    X=10;

    Y=X++; (Incremented value not used in the expression)



Increment and Decrement Operator(++ & --)

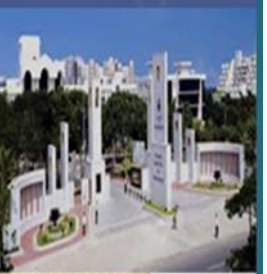
- Decrement Operator(--) → --x , x--
- **Decrements the Value of the variable by 1.**
- **Two Categories**
  - **pre-decrement operator** (or) *prefix decrement operator.*
  - *Operator is before operand – decrements value of the variable first, decremented values is taken for evaluation*

    X=10;

    Y=--X;
  - **post-decrement operator** (or) *postfix decrement operator.*
  - *Operator is after operand – first expression is evaluated and then the value of variable is decremented.*

    X=10;

    Y=X--; (Decremented value not used in the expression)



Conditional or Ternary Operator

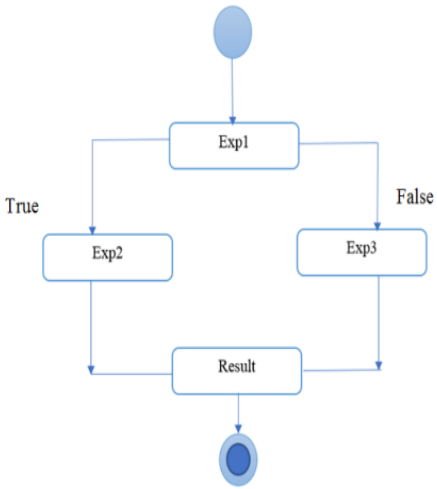
- It works on three operands which is used for decision making.
- Syntax: Condition ? TrueExpression : FalseExpression
- Example:

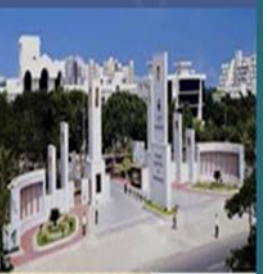
```
a=10;
b=15;
x=(a>b)? a:b
```

or

```
If(a>b)
    x=a;
else
    x=b;
```

- Try Minimum of three numbers.





Special Operators

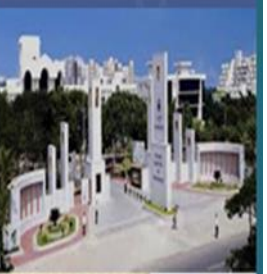
- **Comma**
- **Sizeof**
- Pointer Operators(& and *) - will discuss in module 3
- Member Selection Operators(. and ->) - will discuss in module 4

Comma operator: Used to link the related expressions together.

For example: value = (x=10, y=5, x+5);

Sizeof operator: returns the number of bytes the operand occupies

For Example: m = sizeof(n) or sizeof(float) or sizeof(char)

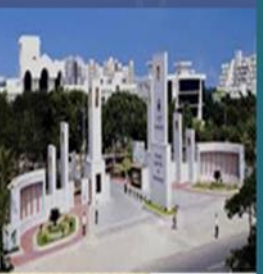


Operator Precedence

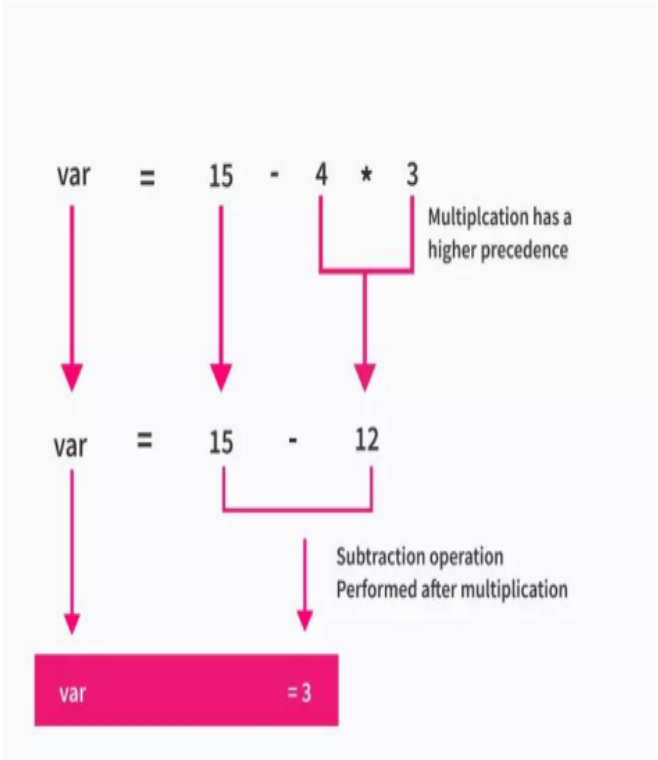
- "C" has precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated.
- Operators at the highest level are evaluated first.
- Operators at the same precedence are evaluated either from left to right or from right to left, depending on the level. This is known as **associativity property of an operator.**
- Example:

```
if (x == 10 + 15 && y < 10)
if (x == 25 && y < 10)
```

- **// Precedence rule:** Addition operator has highest priority than logical and relational operators.



Precedence of Arithmetic Operators



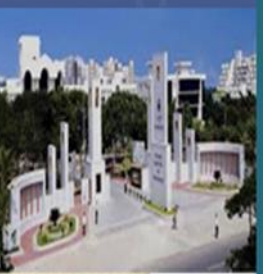
Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ - - (type)* &sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	<< = >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
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

Precedence of Arithmetic Operators

Example :

```
#include <stdio.h>
int main()
{
        float a,b,c,x;
        a=9; b=12; c=3;
        x= a - b / 3 + c * 2 - 1;
        printf("x=%f", x);
        return 0;
}
```

Output ????



Arithmetic Expressions

- **It is a meaningful combination of variables, constants and operators arranged as per the syntax of the language.**

Example: Sum = a + b;

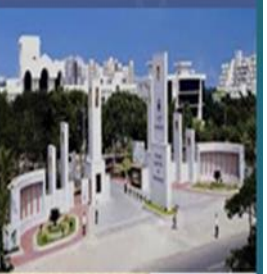
- **SYNTAX :**

<exp> = <exp> op <exp>/<var> op <var>/
<var> op <const>/<const> op <var>/
<const> op <const> /<const>/<var>

where <*op*> may be arithmetic / logical / relational operator

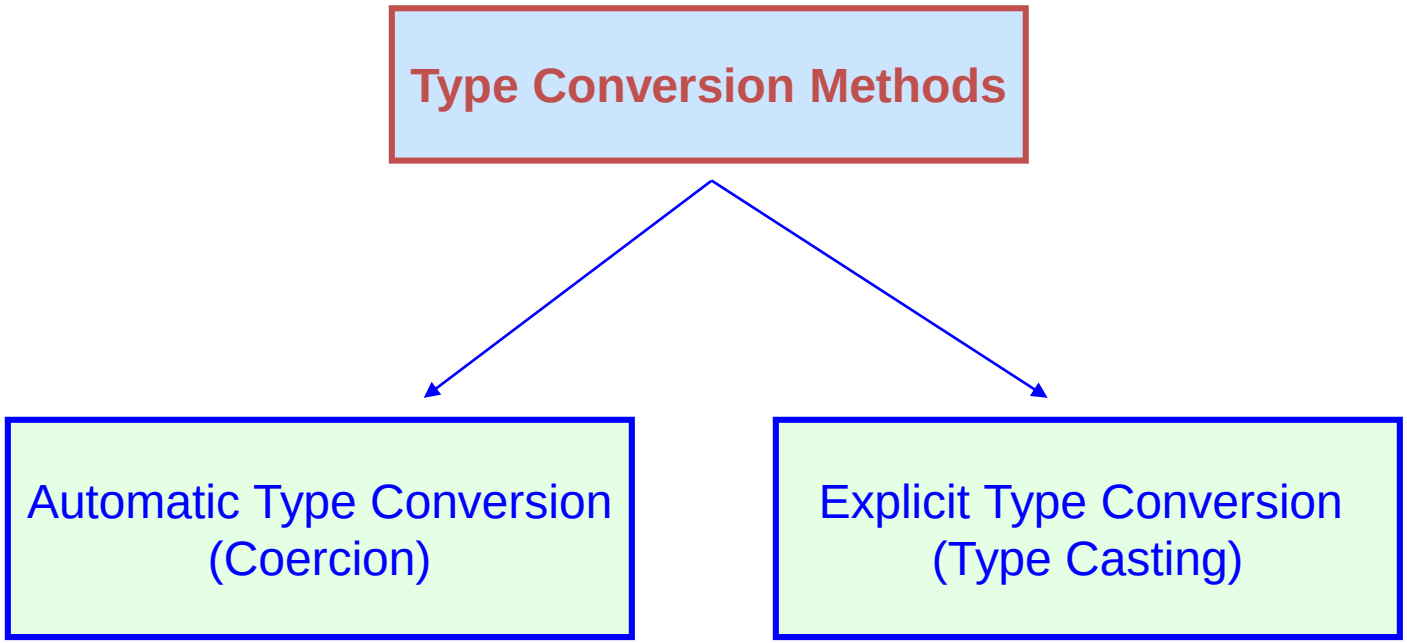
For E.g.

i. Assignment = 10;
ii. Calc = Number + 10;
iii. Choice = 'y'
iv. Interest = (Principal * NoOfYears * RateOfInterest)/100;



Type Conversions in Expressions

- **"C" Permits mixing of constants and variables of different types.**
- "C Converts a variable from one data type to another data type.

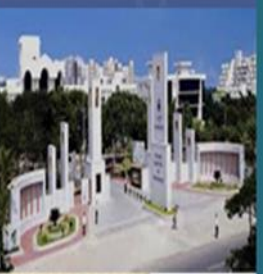


Implicit Type Conversion/Automatic Type

- When the type conversion is performed automatically by the compiler without programmers intervention.
- The compiler converts all operands into the data type of the largest operand.

RULES:

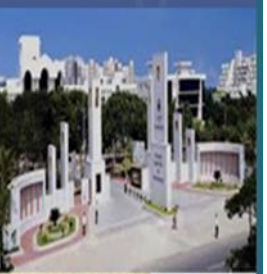
- bool -> char -> short int -> int ->
- unsigned int -> long -> unsigned ->
- long long -> float -> double -> long double
- If either of the operand is of type long double, then others will be converted to long double and result will be long double.
- Else, if either of the operand is double, then others are converted to double.
- Else, if either of the operand is float, then others are converted to float.



Implicit Type Conversions - Example

```
#include<stdio.h>
int main()
{
   int x = 10;    // integer x
   char y = 'a';  // character y
   // y implicitly converted to int. ASCII value of 'a' is 97
   x = x + y;
   // x is implicitly converted to float
   float z = x + 1.0;
   printf("x = %d, z = %f", x, z);
   return 0;
}
```

Output x=??, y=??



Implicit Type Conversions - Example

Example:

```
int number=100;
float conversionValue;
conversionValue=number;

conversionValue = 10.45;
number = conversionValue;
```

No Data Loss

as *small* to *large* conversion 100 will be converted to float as 100.00

Automatic Conversion with Data Loss

Value of number is 10

Explicit Type Conversion

- Type conversion performed by the programmer.
- Explicit type conversion is also known as **type casting**.
- Type casting in c is done in the following form:

(data_type)expression;

- where, *data_type* is any valid c data type, and *expression* may be constant, variable or an expression.
- For example:

```
x=(int)a+b*d;
```

Explicit Type Conversion

```
int numerator=5;
int denominator=2;
float result;
result = numerator/denominator;

result = (float) (numerator/denominator);

result = numerator/(float)denominator;
```

No Type Casting!
The result is 2.0

Wrong Type Casting!
The result is 2.0

Correct Type Casting!
The result is 2.5

Note: Every data type can't be converted into other type

Try Yourself

1. To find the area(Formula= a^2) and perimeter (4 a) of a square.
2. To find the area of a circle($A = \pi r^2$). Use radius as a variable and pi as a constant.
3. To convert Celsius to Fahrenheit and vice-versa.

   C to F= (___* 1.8)+32

   F to C= (___- 32)* 0.5556

1. To convert gram to kilogram and vice-versa.
2. To convert kilometers to miles and vice-versa.
3. To check whether a number is negative, positive or zero.
4. To check input number for odd or even.
5. To check whether a number is divisible by 6 and 8 or not.
6. To find the greatest among three numbers.
7. To check whether the given year is leap year or not.

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Thank You