SLIIT ACADEMY

Higher Diploma in Information Technology
Year 1, Semester 1

Introduction to Programming(C++)

Lecture 03 : Operators in C++

Intended Learning Outcomes

On the Completion of this lecture student will be able to learn ,

LO1 : Understand the working of the operators

LO2: Identify the increment/ decrement operators.

LO3: Evaluate different expressions containing operators.

LO4: Understand about the conditional operators.

LO5: Understand the concept of Type conversion.

Operators in C++

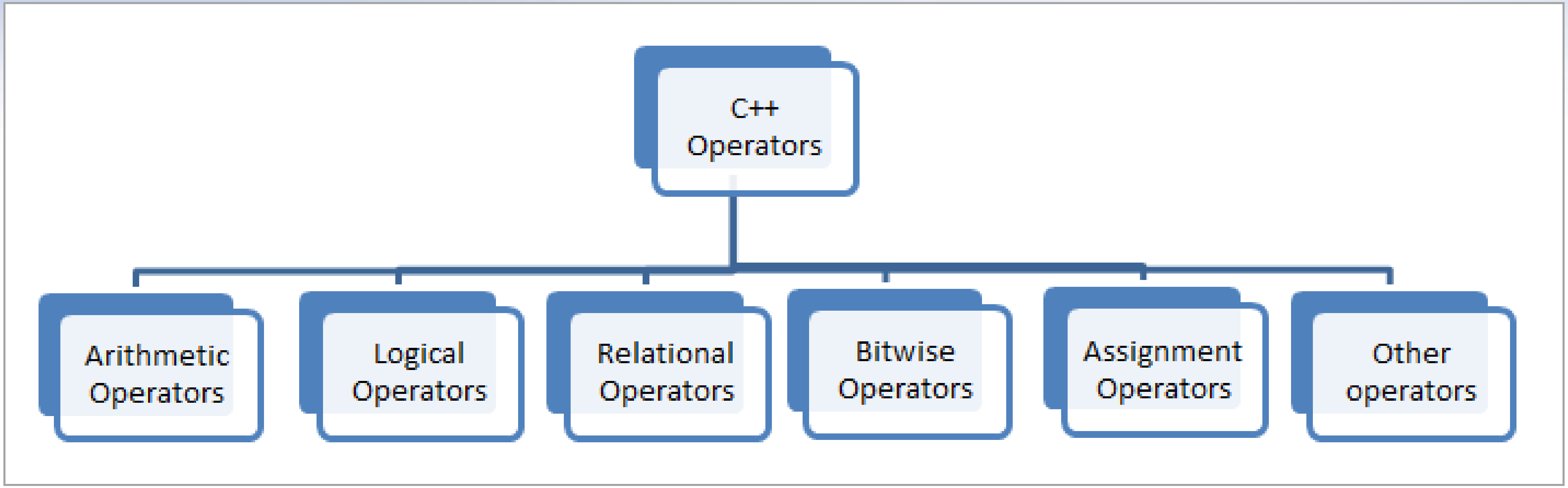
- Operators form the foundation of any programming language.
- Without operators, we cannot modify or manipulate the entities of programming languages and thereby cannot produce the desired results.
- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

Categories of Operators

Type	Explanation	Example
Unary Operator	The operators that operate a single operand to form an expression.	+ + (increment) operator -- (decrement) operator
Binary Operator	The operators that operate two or more operands.	+, -, *, /, %
Ternary Operator	The operators that operates minimum or maximum three operands.	Conditional Operator

SLIIT ACADEMY

Types of Operators in C++



Arithmetic Operators in C++

Operator	Definition
+	Addition
-	Subtraction
*	Multiplication
/	Division (quotient)
%	Division (remainder)
++	Increment
--	Decrement

Integer Division

- Division between two integers results in an integer.
- The result is truncated, not rounded
- Example:
  - 5/3 is equal to 1
  - 3/6 is equal to 0

Increment and Decrement Operators

Increment Operator ++

• post increment	**x++**
• pre increment	**++x**

Decrement Operator - -

• post decrement	**x- -**
• pre decrement	**- -x**

Thumb rule

PREFIX : change and use

CHANGE the Value of the variable

USE the new value

Pre-increment	K=++N	N=N+1
		K=N

Pre-decrement	K=--N	N=N-1
		K=N

POSTFIX : use and change

USE the original value of the variable

CHANGE the Value of the variable

Post-increment	K= N++	K=N
		N=N+1

Post-decrement	K= N--	K=N
		N=N-1

Practice Question 01

What is the final value of **x , y , z**?

```
int x = 5, y = 5, z;
x--; --y;
z = x + y;
```

```
int x = 5, y = 5, z;
++x; --y;
z = x + y++;
```

```
int x = 5, y = 10, z;
z = x++ + --y ;
```

```
int x = 5, y = 10, z;
x++ ; ++y ;
z = --x + y ;
```

```
int x = 5, y = 10, z;
++x ; ++y ;
z = y-- - x++;
```

Priority of Arithmetic and Assignment Operators

Precedence of Arithmetic and Assignment Operators

Precedence	Operator	Associativity
1	Parentheses: **()**	Innermost first
2	Unary operators **++ -- (type)**	Right to left
3	Binary operators *** / %**	Left to right
4	Binary operators **+ -**	Left to right
5	Assignment operators **= += -= *= /= %=**	Right to left

Practice Question 02

Evaluate each of the following expressions and list the final value of variable X.

1. X = 7 + 3 * 5 – 2
2. X = 4 + 7 / 3
3. X= 8 % 3 * 6
4. X =(7 + 3) * 5 – 2
5. X = 16 % 3 - (8 + 5) + 3 / (8 – 6) – 3
6. X = ((10 – 4) + 2) % 8 + (5 + 2) % 3
7. X = 12 – 3 * 4 + 3 + 29 / 5 % 4 * 3 + 6 / 8 % 3
8. X = 15 + 2 % 3 - (8 – 5) + 3 / 8 + 6 % 5
9. X = 17 %(4 + 2) * 8 / (5 + 2) % 3
10. int i=11

    X = i-- % 4 * 4

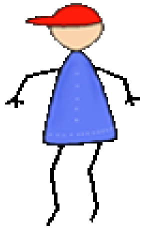
Relational Operators in C++

- A relational operator tests the relationship between two expressions.
- An expression may be a constant, a variable, a function invocation, or a computation involving operators.
- The result of a relational operation is either ***true*** or ***false.***

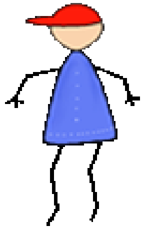
Operator	Definition
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

More about Relational Operators

- A common error is to use the assignment operator instead of the equivalence operator.
- The expression (x = y) assigns the value of y to x and evaluates to ***true*** as long as y was not 0.



= Sam (You are Sam)



== Sam (Are you Sam?)

Logical Operators in C++

Operator		Definition
&&	Logical AND	The compound condition resulting from using the **logical and** operation evaluates as **true** if and only if both expressions are **true.**
||	Logical OR	The compound condition resulting from using the **logical or** operation evaluates as **false** if and only if both expressions are **false**.
!	Logical NOT	The logical not operation negates a logical expression. If the expression originally evaluated to **true**, then the logical not of that expression evaluates to **false**. Likewise, if the expression evaluated as **false** then the logical not of that expression evaluates as **true**.

A logical operator is a symbol that we use to combine or negate expressions that are *true or false*.

Practice Question 03

X	Y	(X < 5)	(Y > = 0)	((X < 5) && (Y > = 0))	((X < 5) || (Y > = 0))
3	0				
2	-2				
9	5				
5	-1				

X	(X>=0)	!(X>=0)	X < 0
5			
-2			

SLIIT
ACADEMY

Practice Question 03

X	Y	(X < 5)	(Y > = 0)	((X < 5) && (Y > = 0))	((X < 5) || (Y > = 0))
3	0	True	True	True	True
2	-2	True	False	False	True
9	5	False	True	False	True
5	-1	False	False	False	False

X	(X>=0)	!(X>=0)	X < 0
5	True	False	False
-2	False	True	True

Bitwise Operators in C++

- Bit by bit operation is performed, and the operator that works on bits is called a bitwise operator.
- Using bitwise operators, there are no byte-level operations in programming; only bit-level calculations are performed in programming. The bits can be manipulated using various bitwise operators.
- The operations of bitwise operators **can be done on integer and character datatypes** only. Bitwise operators **cannot be operated on the float and double.**

Bitwise Operators in C++

1. **& (bitwise AND)** :Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

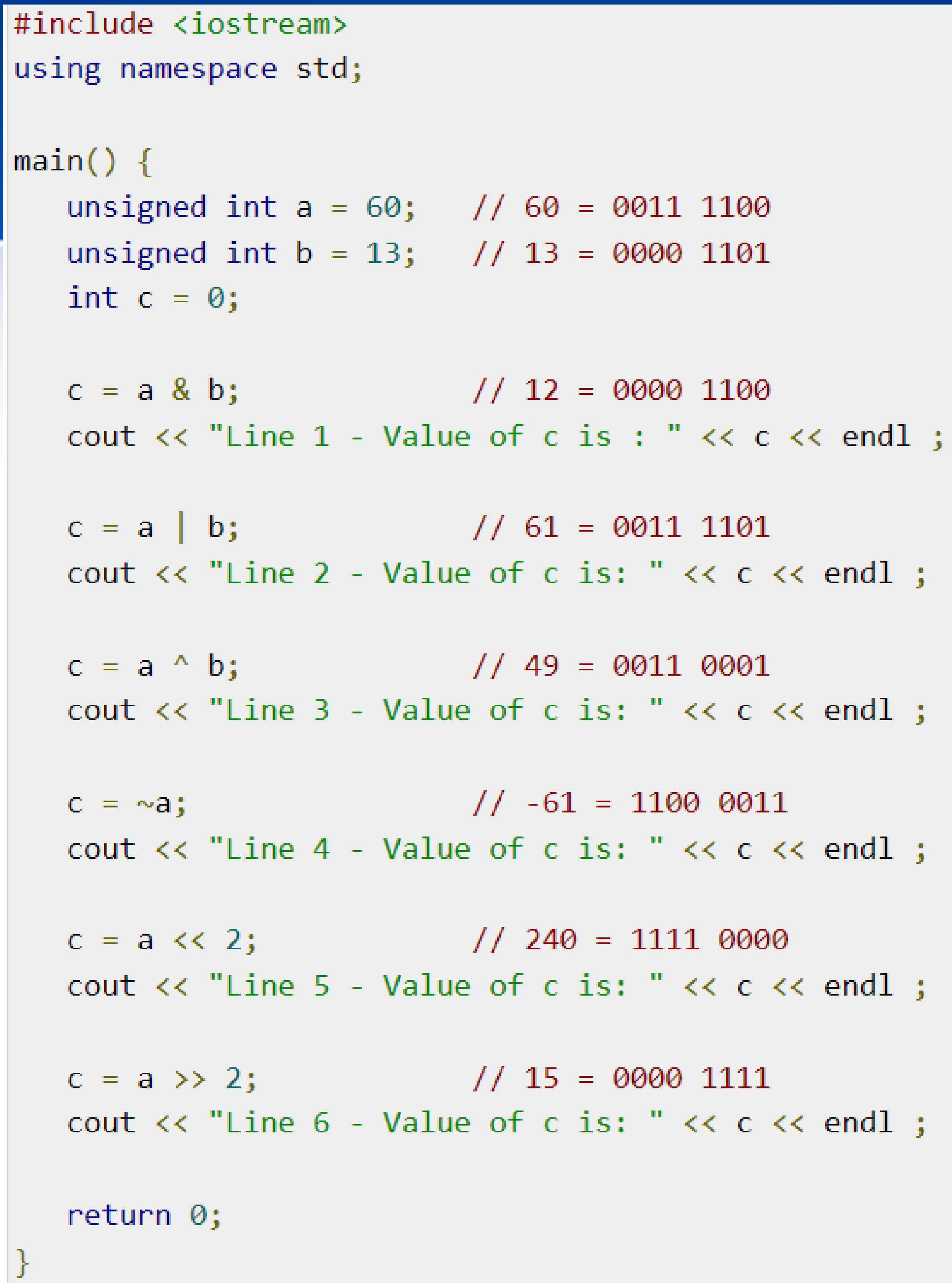
2. **| (bitwise OR)** :Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.

3. **^ (bitwise XOR)** : Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

4. **<< (left shift)** Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.

5. **>> (right shift)** Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

6. **~ (bitwise NOT)** Takes one number and inverts all bits of it.



```cpp
#include <iostream>
using namespace std;

main() {
   unsigned int a = 60;    // 60 = 0011 1100
   unsigned int b = 13;    // 13 = 0000 1101
   int c = 0;

   c = a & b;              // 12 = 0000 1100
   cout << "Line 1 - Value of c is : " << c << endl ;

   c = a | b;              // 61 = 0011 1101
   cout << "Line 2 - Value of c is: " << c << endl ;

   c = a ^ b;              // 49 = 0011 0001
   cout << "Line 3 - Value of c is: " << c << endl ;

   c = ~a;                 // -61 = 1100 0011
   cout << "Line 4 - Value of c is: " << c << endl ;

   c = a << 2;             // 240 = 1111 0000
   cout << "Line 5 - Value of c is: " << c << endl ;

   c = a >> 2;             // 15 = 0000 1111
   cout << "Line 6 - Value of c is: " << c << endl ;

   return 0;
}
```

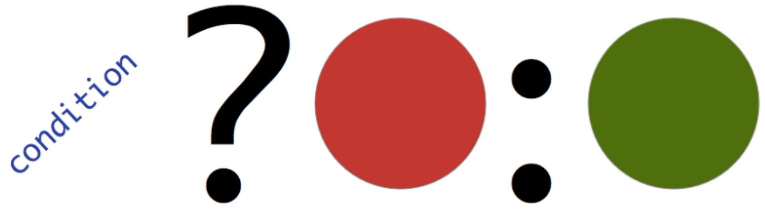
SLIIT ACADEMY

Abbreviated Assignment Operator

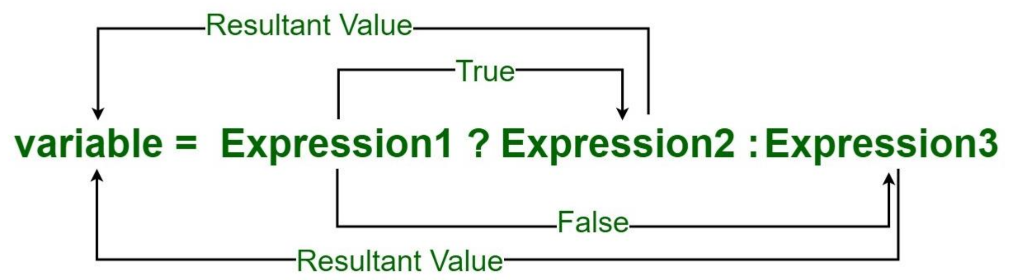
Operator	Example	equivalent statement
+=	x+=2	x=x+2
-=	x-=2	x=x-2
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Conditional Operators

- The conditional operator is a decision-making operator whose statement is evaluated based on the test condition.
- Since the Conditional Operator ‘?:’ takes three operands to work, hence they are also called **ternary operators**.



Conditional Operators



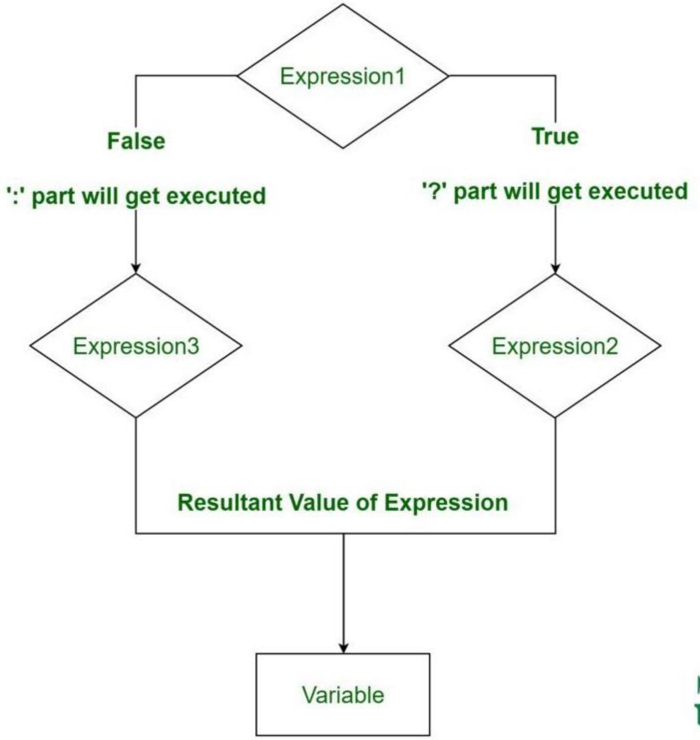
Here, **Expression1** is the condition to be evaluated.

If the condition(**Expression1**) is True then **Expression2** will be executed, and the result will be returned.

Otherwise, if the condition(**Expression1**) is false then **Expression3** will be executed, and the result will be returned

```
Syntax:

variable = Expression1 ? Expression2 : Expression3
```

```
It can be visualized into if-else statement as:
if(Expression1)
{
    variable = Expression2;
}
else
{
    variable = Expression3;
}
```


Expression1
False
True
':' part will get executed
'?' part will get executed
Expression3
Expression2
Resultant Value of Expression
Variable

Practice Question 04

Simple Conditional Operators

```
1. int m = 1, n = 2;
    int min = (m < n ? m : n);
2. int min = (m < n ? m++ : n++);
3. int count=3 , sum = 9
   average = (count == 0) ? 0 : sum / count;
```

Nested Conditional Operators

```
1. int m = 1, n = 2, p =3;
   int min = (m < p ? (m < p ? m : p) : (n < p ? n : p));
```

Comma Operator

- Multiple expressions can be combined into one expression using the comma operator.
- The comma operator takes two operands.
- The comma operator is a special operator ***which evaluates statements from left to right and returns the rightmost expression as the final result***

Practice Question 05

What is the final value of **c** ?

```
int a = 1, b = 2, c;

c = (a = a + 2, b = a + 3, b = a + b);
```

What is the final value of **y** ?

```
int x = 10 ,y;

y = (x++, ++x);
```

What is the final value of **k and x**?

```
int x = 5 ,x;

k = (x--, x--);
```

sizeof Operator

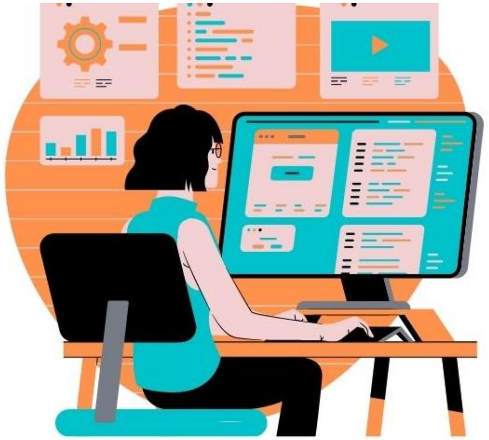
- The sizeof operator determines **the storage size of a particular value, type, or variable in bytes.**

Syntax

```
sizeof(item)
```

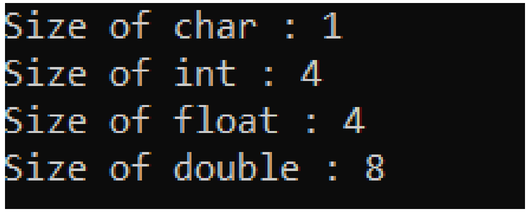
Example

```
sizeof(long int)
sizeof(num)
sizeof(“Today”)
sizeof(5.29)
```



sizeof Operator

```
#include <iostream>
using namespace std;
int main() {
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   return 0;
}
```



```
Size of char : 1
Size of int : 4
Size of float : 4
Size of double : 8
```

Type Casting / Conversion in C++

- Type casting refers to the conversion of one data type to another in a program.
- Typecasting can be done in two ways: **automatically by the compiler and manually by the programmer or user.**
- Type Casting is also known as **Type Conversion.**
- Type Casting is divided into two types:
  - **Implicit Type Casting**
  - **Explicit Type Casting.**

SLIIT ACADEMY

Implicit Type Conversion

- Implicit type conversion also known as **automatic type conversion.**
- It automatically converted from one data type to another without any external intervention such as programmer or user. It means the compiler automatically converts one data type to another.
- All data type is automatically upgraded to the largest type without losing any information.
- It can only apply in a program if both variables are compatible with each other.

Implicit Type Conversion

- The order of the automatic type conversion is listed below:

bool -> char -> short int -> int ->

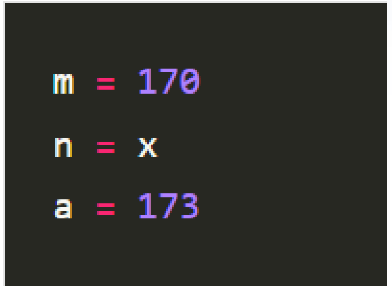
unsigned int -> long -> unsigned ->

long long -> float -> double -> long double

Implicit Type Conversion

```
#include <iostream>
using namespace std;
int main()
{
        int m = 50; // integer m
        char n = 'x'; // character n

        // n is implicitly converted to int. ASCII value of 'x' is 120
        m = m + n;

        // m is implicitly converted to float
        float a = m + 3.0;

        cout << "m = " << m << "n = " << n << "a = " << a << endl;
        return 0;
}
```

Output:



```
m = 170
n = x
a = 173
```

Explicit Type Conversion

- It is also known as the **manual type casting** in a program.
- It is manually cast by the programmer or user to **change from one data type to another type in a program**. It means a user can easily cast one data to another according to the requirement in a program.
- It does not require checking the compatibility of the variables.
- In this casting, we can upgrade or downgrade the data type of one variable to another in a program.
- It uses the cast () operator to change the type of a variable.

Explicit Type Conversion

Syntax of the explicit type casting

```
(type) expression;
```

- type: It represents the user-defined data that converts the given expression.
- expression: It represents the constant value, variable, or an expression whose data type is converted.

- Example : we have a floating pointing number is 4.534, and to convert an integer value, the statement as:

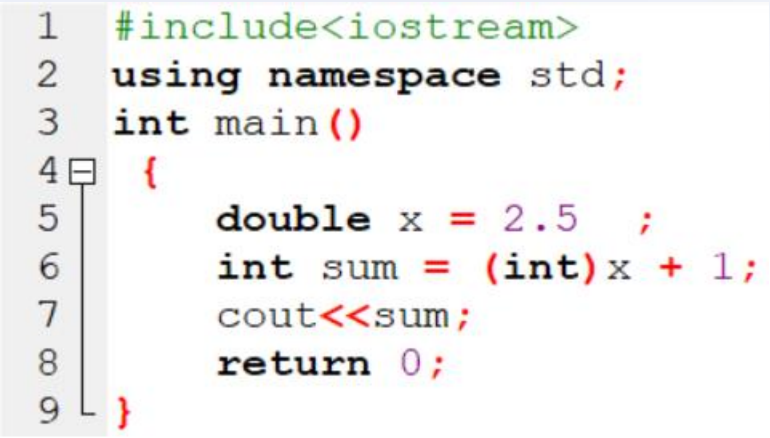
```
int num;
num = (int) 4.534; // cast into int data type
cout << num;
```

Implicit & Explicit Type Conversion

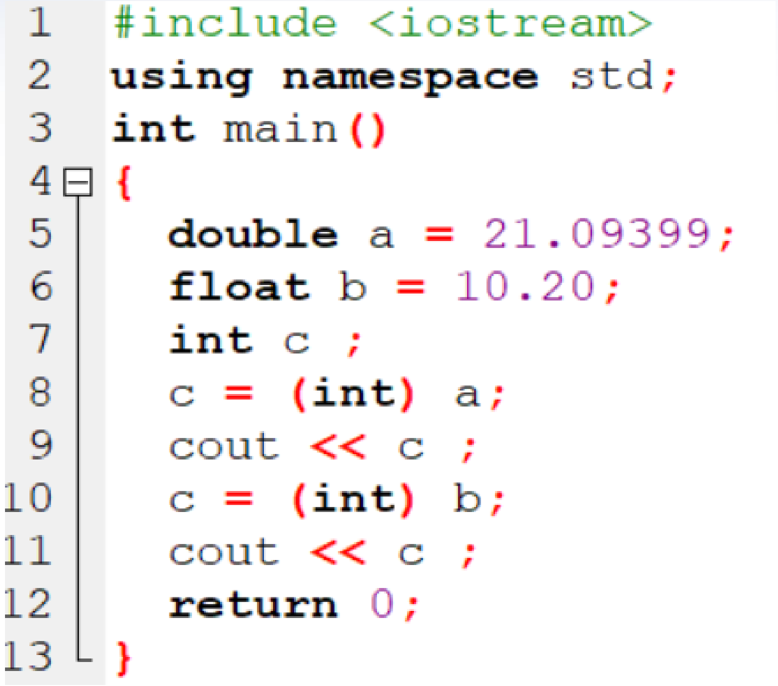
```
#include <iostream>
using namespace std;
int main ()  {
    // declaration of the variables
    int a=21, b=5;
    float res;
    cout << " Implicit Type Casting: " << endl;
    cout << " Result: " << a / b << endl; // it loses some information

    cout << " \n Explicit Type Casting: " << endl;
    // use cast () operator to convert int data to float
    res = (float) 21 / 5;
    cout << " The value of float variable (res): " << res << endl;
    return 0; }
```

Practice Question 06

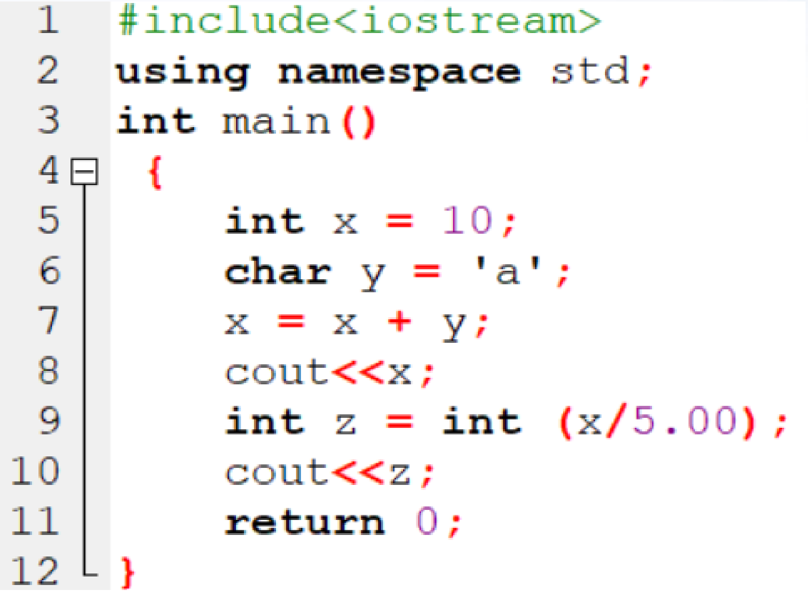
What will be the output(s) of the following C++ codes?



```cpp
#include<iostream>
using namespace std;
int main()
 {
     double x = 2.5   ;
     int sum = (int)x + 1;
     cout<<sum;
     return 0;
}
```



```cpp
#include <iostream>
using namespace std;
int main()
{
   double a = 21.09399;
   float b = 10.20;
   int c ;
   c = (int) a;
   cout << c ;
   c = (int) b;
   cout << c ;
   return 0;
}
```



```cpp
#include<iostream>
using namespace std;
int main()
 {
     int x = 10;
     char y = 'a';
     x = x + y;
     cout<<x;
     int z = int (x/5.00);
     cout<<z;
     return 0;
}
```

Mathematical functions in C++ - The cmath library

C++ has many functions that allows you to perform mathematical tasks on numbers.

abs(x)	computes absolute value of x
sqrt(x)	computes square root of x, where x >=0
pow(x,y)	computes x^y
ceil(x)	nearest integer larger than x
floor(x)	nearest integer smaller than x
exp(x)	computes e^x
log(x)	computes ln x, where x >0
log10(x)	computes $\log_{10}$x, where x>0
sin(x)	sine of x, where x is in radians
cos(x)	cosine of x, where x is in radians
tan(x)	tangent of x, where x is in radians

Practice Question 07

Write a C++ program(s) to calculate the followings:

- Take two integer numbers as user inputs and find the maximum and minimum among the numbers.
- Take one number as user input and print the square root of the entered number.
- Take two integer numbers as user inputs . Consider one number as base and other as power. Calculate the answer as base to the power.
- Take the fractional number as an input and find out the ceiling value and floor value of entered number.

Hint : Use the suitable mathematical functions.

Summary

- Introduction to operators in C++
- Categories of operators in C++
- Arithmetic operators in C++
- Relational operators in C++
- Logical operators in C++
- Conditional operators in C++
- Type conversion in C++
- Mathematical functions in C++