Fundamentals of Programming: Operators

Abdul Haseeb

Agenda

- Operators
- Arithmetic Operators
- Incremental or Decremental Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Shift Operator
- Assignment Operator
- Ternary Operator

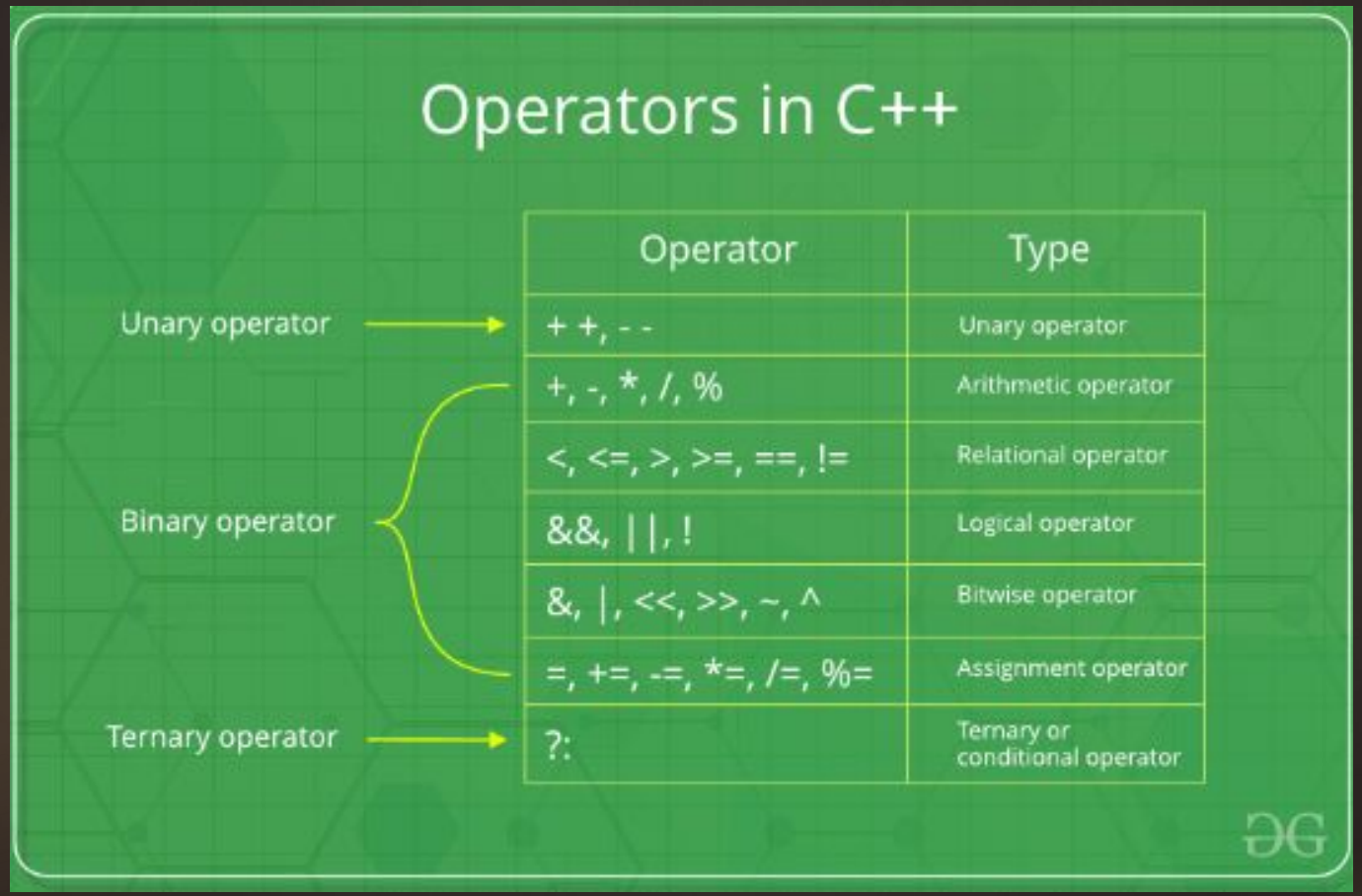
Introduction

- Operators perform operations on operands
- Operands are variables on which operation is performed

```
int c = a + b;
```

Here, '+' is the addition operator. 'a' and 'b' are the operands that are being 'added'.

Introduction



Arithmetic Operators

- Perform arithmetic or mathematical operations on the operands
- Arithmetic operators can be classified into two categories:
  - Unary {Increment operator(++), Decrement Operator (--)}
  - Binary(+, -, *, /, %)

Binary Arithmetic Operators

- Work on two operands

Name	Symbol	Description	Example
Addition	+	Adds two operands	int a = 3, b = 6; int c = a+b; // c = 9
Subtraction	–	Subtracts second operand from the first	int a = 9, b = 6; int c = a-b; // c = 3
Multiplication	*	Multiplies two operands	int a = 3, b = 6; int c = a*b; // c = 18
Division	/	Divides first operand by the second operand	int a = 12, b = 6; int c = a/b; // c = 2
Modulo Operation	%	Returns the remainder an integer division	int a = 8, b = 6; int c = a%b; // c = 2

Example

```cpp
// CPP Program to demonstrate the Binary Operators
#include <iostream>
using namespace std;

int main()
{
    int a = 8, b = 3;

    // Addition operator
    cout << "a + b = " << (a + b) << endl;

    // Subtraction operator
    cout << "a - b = " << (a - b) << endl;

    // Multiplication operator
    cout << "a * b = " << (a * b) << endl;

    // Division operator
    cout << "a / b = " << (a / b) << endl;

    // Modulo operator
    cout << "a % b = " << (a % b) << endl;

    return 0;
}
```

Output

```
a + b = 11
a - b = 5
```

String Concatenation

String Concatenation

String concatenation is the act of combining two strings together. This is done with the + operator.

```
string a = "This is an ";
string b = "example string";
string c = a + b;
cout << c << endl;
```

challenge

What happens if you:

- Concatenate two strings without an extra space (e.g. remove the space after an in `string a = "This is an";`)?
- Use the += operator instead of the + operator (e.g. `a+=b` instead of `a + b`)?
- Add `3` to a string (e.g. `string c = a + b + 3;`)?
- Add `"3"` to a string (e.g. `string c = a + b + "3";`)?

Trick your Brain with 20 mint rule

Unary Operator

- **Work or operate on a single operand.**

Name	Symbol	Description	Example
Increment Operator	++	Increases the integer value of the variable by one	int a = 5; a++; // returns 6
Decrement Operator	—	Decreases the integer value of the variable by one	int a = 5; a–; // returns 4

Pre-Increment vs Post-Increment

- **Pre-increment:**
  - Incremented value of variable is used in expression
  - int a=5;
  - cout<<++a;
- **Post-increment:**
  - Current value of variable is used in expression, and after that value in incremented.
  - int a=5;
  - int b=a++;
  - cout<<b;
  - Let’s try to access a.

Pre-Decrement vs Post-Decrement

- **Pre-Decrement:**
  - Decremented value of variable is used in expression
  - int a=5;
  - cout<<--a;
- **Post-Decrement:**
  - Current value of variable is used in expression, and after that value in Decremented.
  - int a=5;
  - int b=a--;
  - cout<<b;
  - Let’s try to access a.

Short hand Assignment

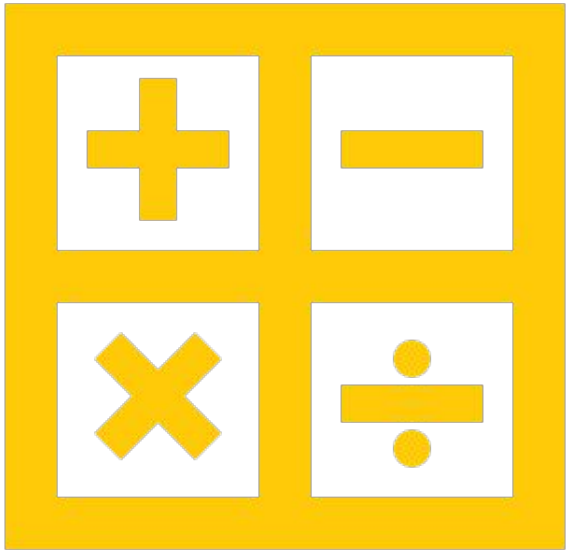
- int count=2;
- count+=2; //Equivalent to writing count=count+2;
- count*=2; //Equivalent to writing count=count*2;
- count-=2; //Equivalent to writing count=count-2;
- count/=2 //Equivalent to writing count=count/2;
- count%=2 ////Equivalent to writing count=count%2;

Type Conversion(Casting)

- The process of converting one data type to another
- Two Types:
  - Implicit (Done by Compiler/Automatic)
  - Explicit (Done by Programmer/Manual)

When is the casting actually performed?

- Arithmetic operations are normally performed over the same types of operands
- But when we have operands of different data, like one operand is character and other one is integer
  - C++ will convert the one operand to be the type of other and then evaluate the expression



IMPLICIT TYPE CONVERSION

- The Data Type conversion that is done by compiler automatically
  - 1. Automatic (Lower to Higher)

Lower ⟶ Higher

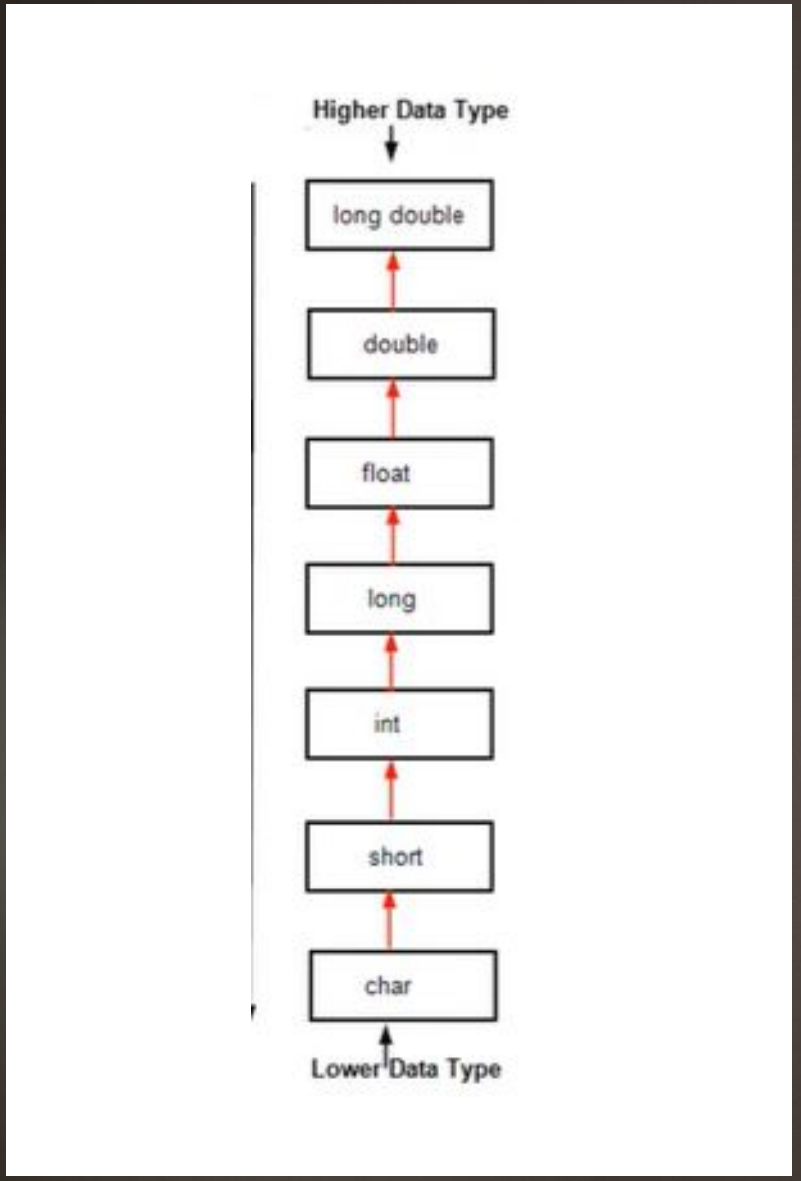
Char, short int	**int**	**float**	**double**	**long double**

  - 2. By Assignment (Right to Left)

IMPLICIT TYPE CONVERSION(LOWER TO HIGHER)

- 2+5.6+9 so what should be the resultant data type?
- 2.0+5.6+9.6= DOUBLE TYPE

- ‘a’+1, What should be the result?
- Result will be 98



Lower to Higher

Check data type of a variable or value

- #include<typeinfo>
- typeid(variable/expression).name()

- Example:
- cout<<typeid(5.9+6).name();

IMPLICIT CONVERSION RIGHT TO LEFT

- float a=12.5;
- int b=13;
- int sum=a+b;
- What will be the result of sum?
- First Lower to Higher, then Left to Right
- a+b=12.5+13.0=25.5 (which is a float value)
- sum=25 (Hence we loose information)

IMPLICIT CONVERSION RIGHT TO LEFT

- char c1=‘a’+1;
- char c1=97+1;
- char c1=98;
- c1 will have value of b because of left to right conversion
- What if we write int c1=‘a’;?
- Obviously we will get 97

EXPLICIT TYPE CONVERSION

- The Type of conversion that you as a programmer specify and you want to do.

- char c1=(char)97
- float f1=(float)9
- cout<<(double)5.3/4

Second method for explicit casting


```
#include <iostream>
using namespace std;
int main()
{
    float f = 3.5;

    // using cast operator
    int b = static_cast<int>(f);

    cout << b;
}
```

Relational Operators

- C++ Relational operators specify the relation between two variables by comparing them.
- If the results after comparison b/w two variable is true it will return 1, else it will return 0 for false.
- There are six relational operators:
  - Less than (<)
  - Less than or equal to (<=)
  - Greater than (>)
  - Greater than or equal to (>=)
  - Equals Equals to (==)
  - Not equal to (!=)

```
1  #include <iostream>
2  using namespace std ;
3  int main ()
4  {
5
6      cout <<"10 > 100     : "     << (10 > 100) <<endl ;
7      cout <<"20 >= 20     : "     << (20 >= 20) <<endl ;
8      cout <<"10 < 100     : "     << (10 < 100) <<endl ;
9      cout <<"30 <= 40     : "     << (30 <= 40) <<endl ;
10     cout <<"30 != 30     : "     << (30 != 30) <<endl ;
11     cout <<"40 == 30     : "     << (40 == 30) <<endl ;
12
13     system ("PAUSE") ;
14     return 0 ;
15 }
```

```
10 > 100          : 0
20 >= 20          : 1
10 < 100          : 1
30 <= 40          : 1
30 != 30          : 0
40 == 30          : 0
```

Operators	Name of the Operator	Type
&&	AND Operator	Binary
||	OR Operator	Binary
!	NOT Operator	Unary

Logical Operators

LOGICAL OPERATORS ARE USED IF WE WANT TO COMPARE MORE THAN ONE CONDITION.

Logical Operators

Operator	Output
AND	Output is 1 only when conditions on both sides of Operator become True
OR	Output is 0 only when conditions on both sides of Operator become False
NOT	It gives inverted Output

Logical Operators

AND AND (&&) Logical Operator

Condition 1	Condition 2	Overall Results
0	0	0
0	1	0
1	0	0
1	1	1

OR OR (||) Logical Operator

Condition 1	Condition 2	Overall Results
0	0	0
0	1	1
1	0	1
1	1	1

OR OR (||) Logical Operator

```
!(0) = true or 1
! (1) = false or 0
```

Logical Operators

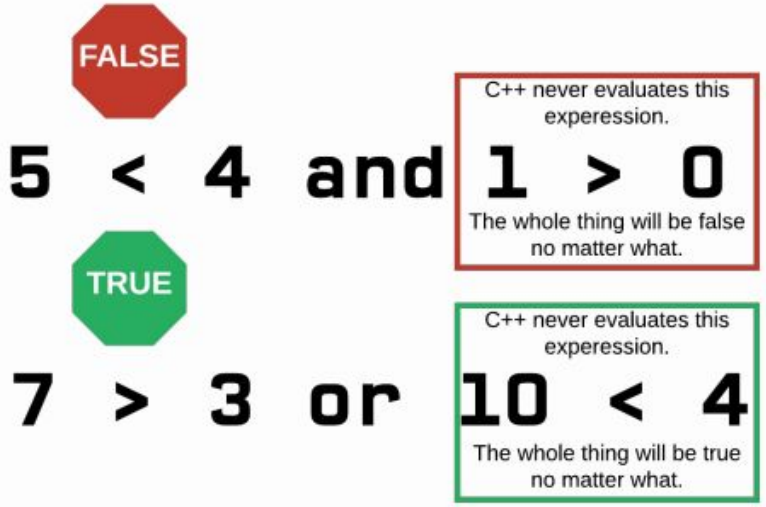
```
cout <<((10 >= 20) && (10 == 10))<<endl ;
cout <<((10 >= 20) || (10 == 10))<<endl ;
cout <<(!(10 <= 20) || !(10 == 10))<<endl ;
```

```
0
1
0
```

Short Circuiting

Short Circuiting

If C++ can determine the result of a boolean expression before evaluating the entire thing, it will stop and return the value.



Short Circuiting

Assignment Operators

- To assign the values to variables
- Assignment Operator is denoted by equal to (=) sign.
- This operator copies the value at the right side of the operator into the left side variable.
- Assignment Operator is binary operator.
- In this example, 10 is assigned to variable named value.

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

int main()
{
    int value;
    value=10;
    return 0;
}
```

Bitwise Operators

- Operate on the individual data bit.
- C++ Bitwise Operators operate on Integer and character data types only.
- C++ Bitwise Operators do not operate on float, double.
- There are four bitwise operators

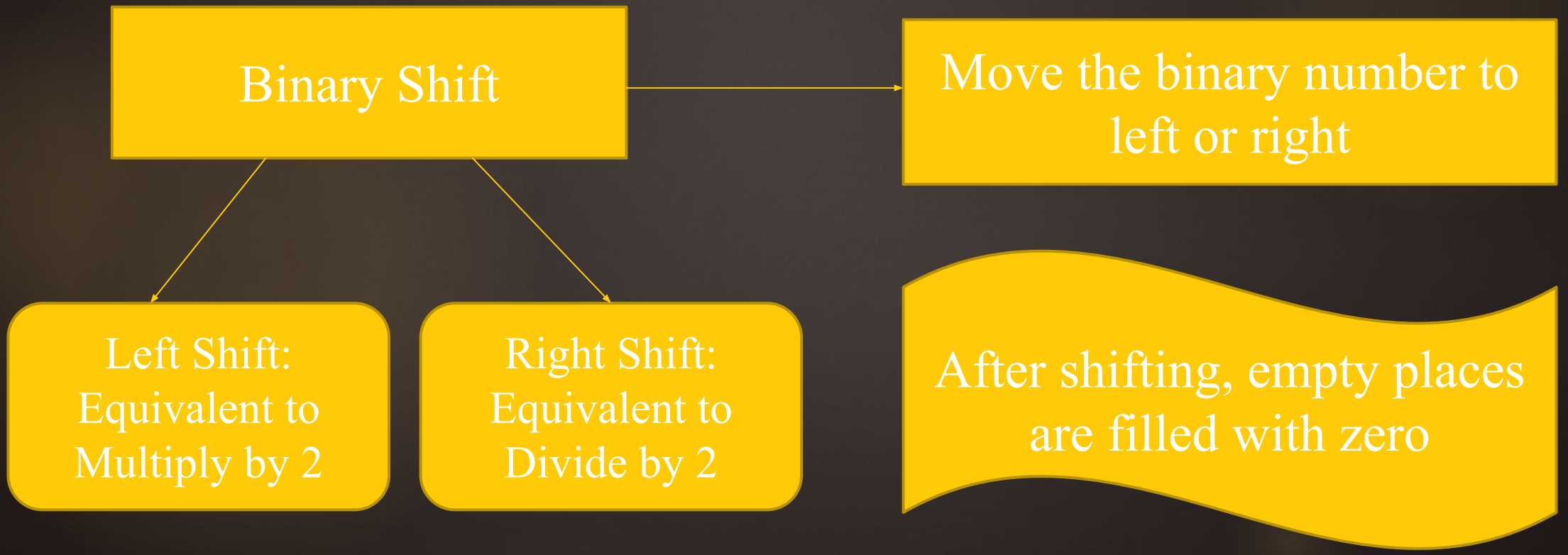
1. Bitwise AND (&)
2. Bitwise OR (|)
3. Bitwise XOR (^)
4. Bitwise One's Compliment (~)

```
=========================================================
        16      8       4       2       1
=========================================================
10 =    0       1       0       1       0
20 =    1       0       1       0       0
=========================================================

& =     0       0       0       0       0 = 0
| =     1       1       1       1       0 = 30
^ =     1       1       1       1       0 = 30

~(10) = -11
~(-20) = 19
```

```
1  #include <iostream>
2  using namespace std ;
3  int main ()
4  {
5
6      cout << (10 & 20) <<endl ;
7      cout << (10 | 20) <<endl ;
8      cout << (10 ^ 20) <<endl ;
9      cout << (~10) <<endl ;
10
11     system ("PAUSE") ;
12     return 0 ;
13 }
14
```

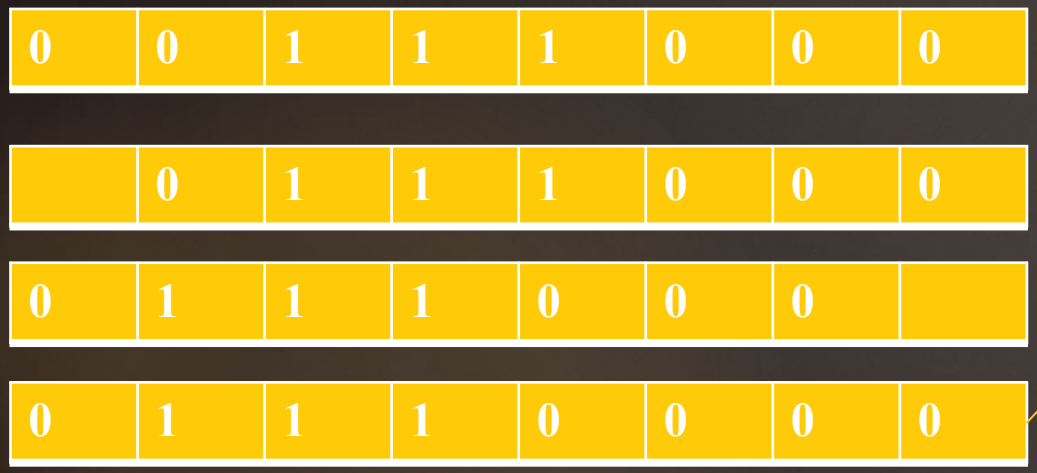
```
0
30
30
-11
```

LOGICAL BINARY SHIFTS



LEFT SHIFT

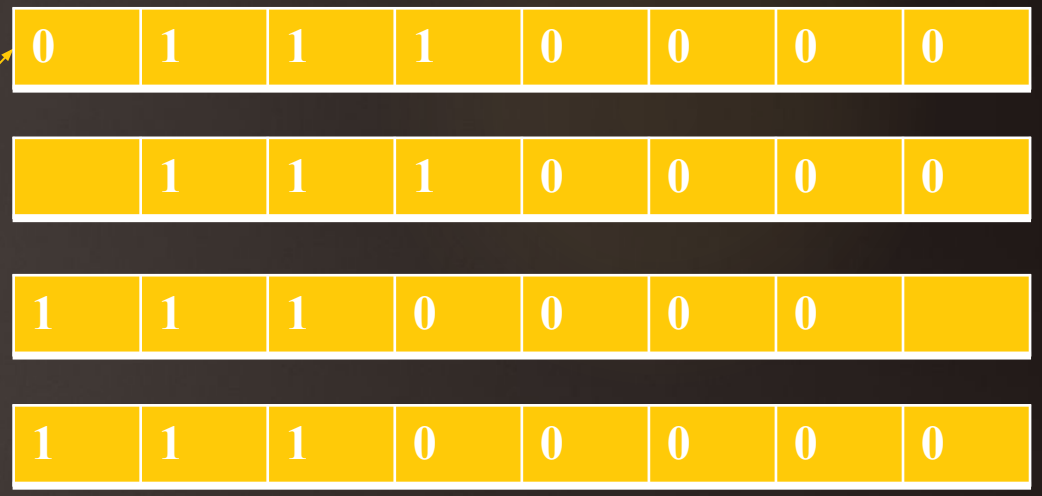
Perform Left Shifting Two places to the left



First Left Shift Done

Denary of Original Number is: 56

What should be the denary after first left shift?

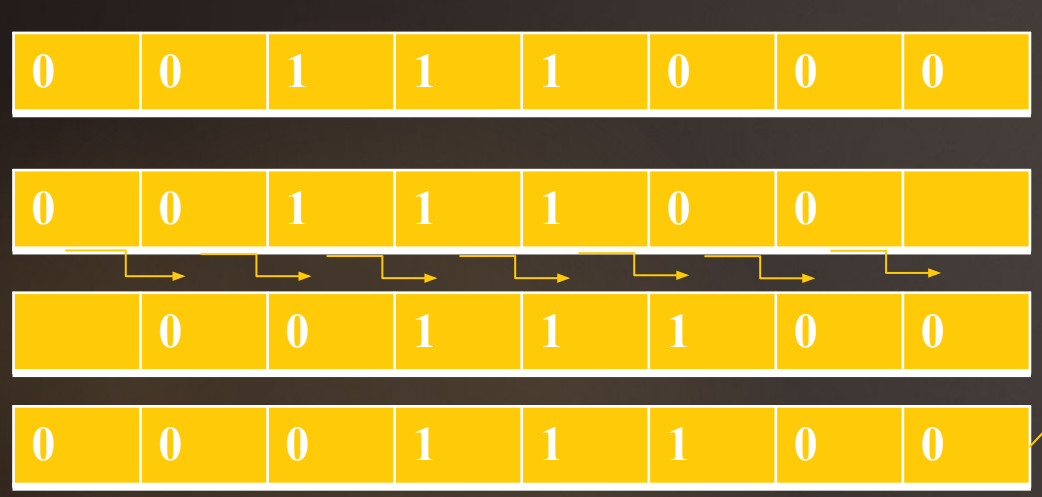


Second Left Shift Done

Denary After Second Right Shift will be: $56*2^{2}=224$

RIGHT SHIFT

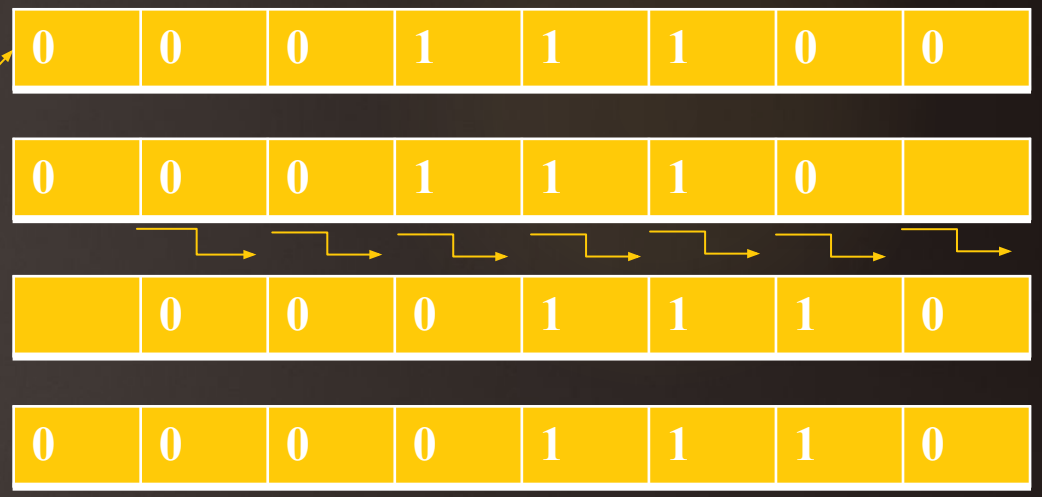
Perform RIGHT Shifting Two places to the RIGHT



First Right Shift Done

Denary of Original Number is: 56

What should be the denary after first Right shift?



Second Left Shift Done

Denary After Second Right Shift will be: $56/2^{2=}14$

Operator precedence and associativity

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right →
2	`a++` `a--`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`.` `->`	Member access	
3	`++a` `--a`	Prefix increment and decrement	Right-to-left ←
	`+a` `-a`	Unary plus and minus	
	`!` `~`	Logical NOT and bitwise NOT	
	(*type*)	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`co_await`	await-expression (C++20)	
	`new` `new[]`	Dynamic memory allocation	
	`delete` `delete[]`	Dynamic memory deallocation	
4	`.*` `->*`	Pointer-to-member	Left-to-right →
5	`a*b` `a/b` `a%b`	Multiplication, division, and remainder	
6	`a+b` `a-b`	Addition and subtraction	
7	`<<` `>>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator (since C++20)	
9	`<` `<=` `>` `>=`	For relational operators < and ≤ and > and ≥ respectively	
10	`==` `!=`	For equality operators = and ≠ respectively	
11	`a&b`	Bitwise AND	
12	`^`	Bitwise XOR (exclusive or)	
13	`|`	Bitwise OR (inclusive or)	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional[note 2]	Right-to-left ←
	`throw`	throw operator	
	`co_yield`	yield-expression (C++20)	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+=` `-=`	Compound assignment by sum and difference	
	`*=` `/=` `%=`	Compound assignment by product, quotient, and remainder	
	`<<=` `>>=`	Compound assignment by bitwise left shift and right shift	
	`&=` `^=` `|=`	Compound assignment by bitwise AND, XOR, and OR	
17	`,`	Comma	Left-to-right →

Ternary Operators

- The ternary or conditional operator is an operator used in C++.
- Sign is **?:**
- This operator returns one of two values depending on the result of an expression.

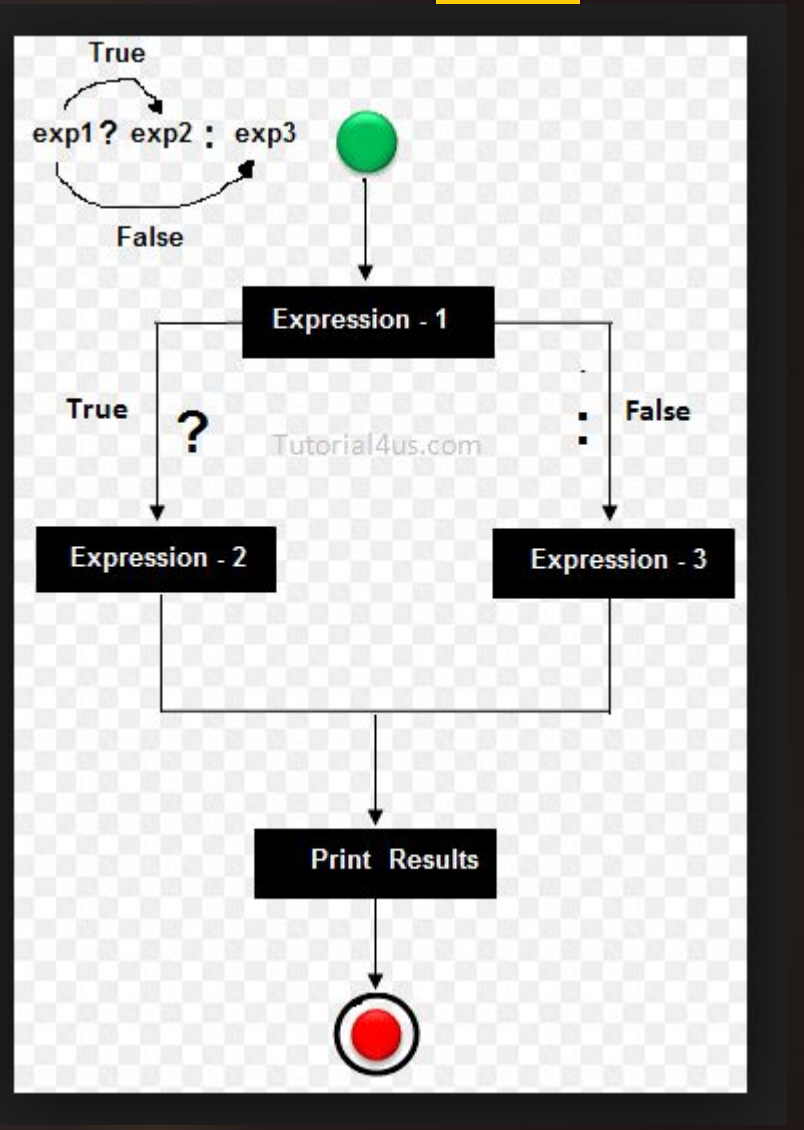
Syntax

```
(expression 1) ? expression 2 : expression 3
```

If *expression 1* evaluates to true, then *expression 2* is evaluated.

If *expression 1* evaluates to false, then *expression 3* is evaluated instead.

(condition) ? (if_true) : (if_false)



Ternary Operators


```
int num1 ;
int num2 ;
cout <<"Enter number 1 : " ; cin >> num1 ;
cout <<"Enter number 2 : " ; cin >> num2 ;
cout <<"The larger number b/w num1 and num is : " ;
cout << ((num1 > num2) ? (num1) : (num2)) <<endl;
```

```
Enter number 1 : 50
Enter number 2 : 100
The larger number b/w num1 and num is : 100
```

```
Enter number 1 : 100
Enter number 2 : 50
The larger number b/w num1 and num is : 100
```

Ternary Operators


```cpp
#include <iostream>
using namespace std ;
int main ()
{
    int num ;
    cout << "Enter any number : " ; cin >> num ;
    string res = ((num % 2 == 0) ? ("it is an even number") : ("It is an odd number")) ;
    cout <<res << endl  ;
    return 0 ;
}
```