Basic Computer Programming and Networking

Operators

RHNS Jayathissa

Department of Computer Engineering

Faculty Of Computing

Example

What is the output of the following program

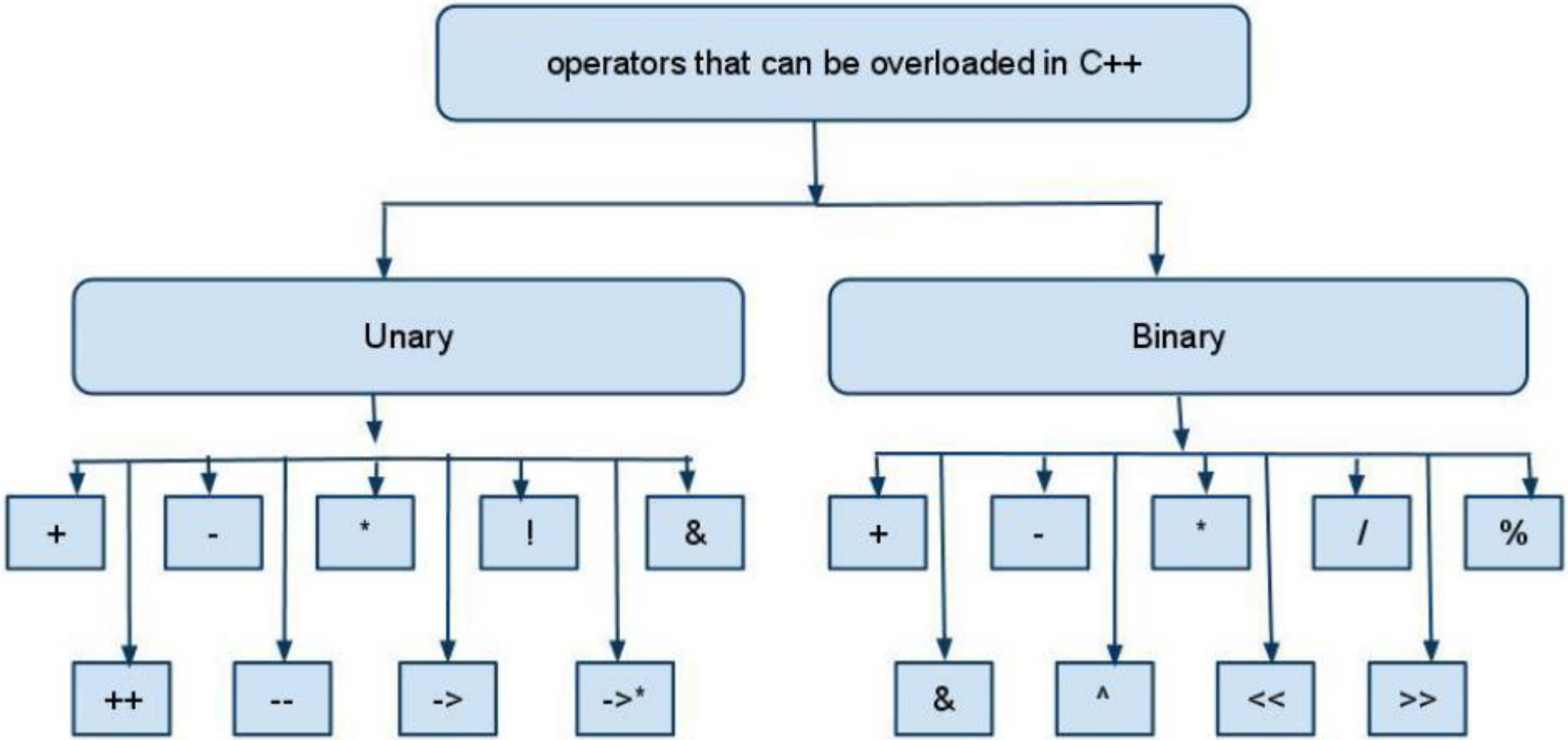
```
int main()
{
      int a = 5;
      int b = 3;
      int c = 2;
      float Y = Y = -(b + c) / a;
      cout << "Y is " <<Y << endl;
      return 0;
}
```

C++ Operators

- Assignment operator
  - (=)
- Arithmetic operators
  - (+, -, *, /, %)
- Compound assignment
  - (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)
- Increment and decrement
  - (++, --)
- Relational and comparison operators
  - (==, !=, >, <, >=, <=)
- Logical operators
  - (!, &&, ||)
- Conditional ternary operator
  - (?)
- Comma operator
  - (,)

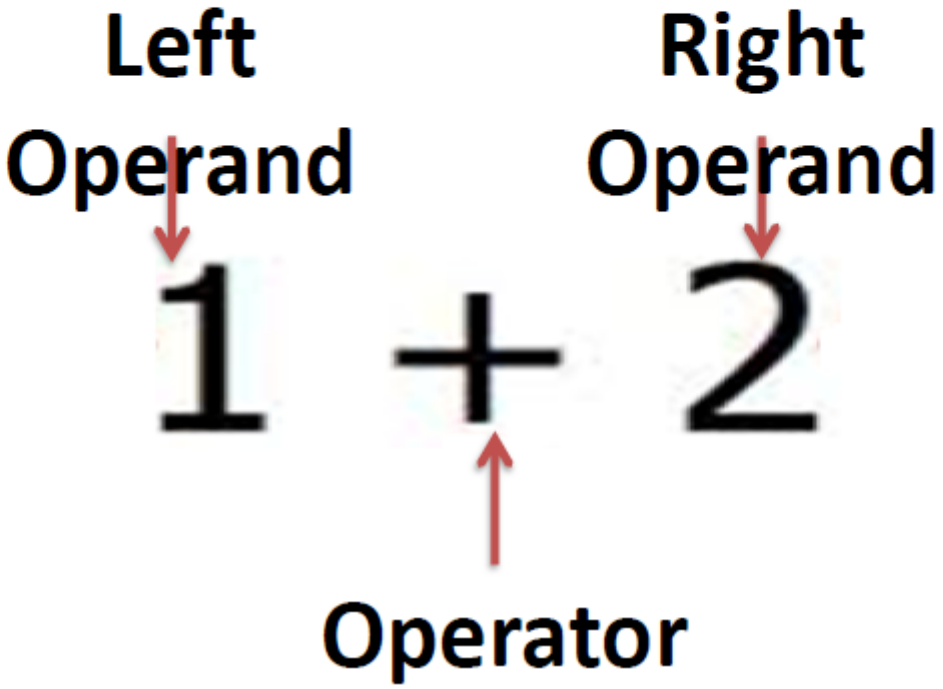
Shorthand operator	Meaning
x += y	x = x + y
x -= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

C++ Operators



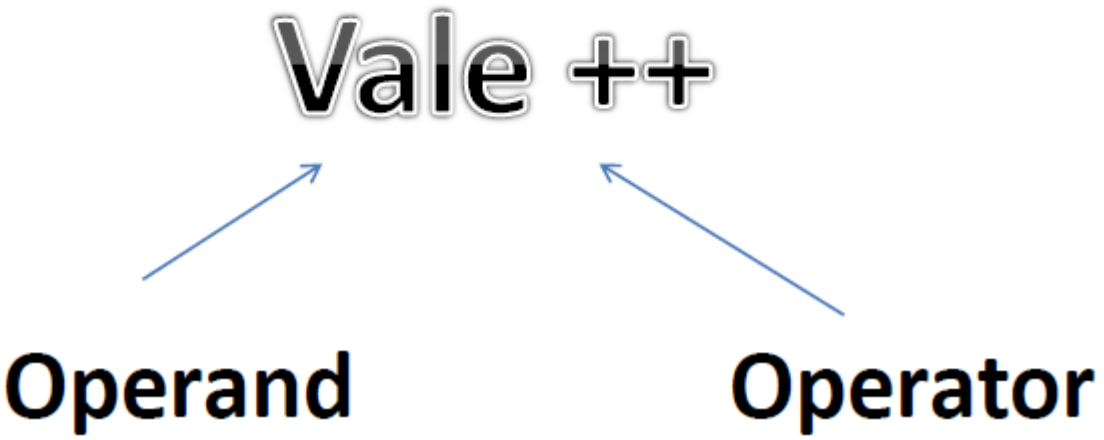
Binary Operators

- The binary operators take two arguments as operands



Unary Operators

The unary operators take one arguments as operand



Assignment operator (=)

- The assignment operator assigns a value to a variable.

```
// assignment operator
#include <iostream>
using namespace std;

int main ()
{
  int a, b;         // a:?,  b:?
  a = 10;           // a:10, b:?
  b = 4;            // a:10, b:4
  a = b;            // a:4,  b:4
  b = 7;            // a:4,  b:7

  cout << "a:";
  cout << a;
  cout << " b:";
  cout << b;
}
```

Arithmetic operators(+, -, *, /, %)

- The five arithmetical operations supported by C++ are

operator	description
+	addition
–	subtraction
*	multiplication
/	division
%	modulo

Compound assignment

(+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

expression	equivalent to...
`y += x;`	`y = y + x;`
`x -= 5;`	`x = x - 5;`
`x /= y;`	`x = x / y;`
`price *= units + 1;`	`price = price * (units+1);`

Combined assignment operators

- Each arithmetic operator has a corresponding assignment operator.

Operator	Effect L = Left Operator R = Right Operator
+=	Assign (L + R) to L
-=	Assign (L - R) to L
*=	Assign (L * R) to L
/=	Assign (L / R) to L
%=	Assign (L % R) to L

Example

```
// compound assignment operators
#include <iostream>
using namespace std;

int main ()
{
  int a, b=3;
  a = b;
  a+=2;                    // equivalent to a=a+2
  cout << a;
}
```

Unary Operators

The unary operators take one arguments

-	unary minus (negation)
+	unary plus
--	decrement
++	increment

Unary Operators

- The unary minus (-) makes a positive number into a negative number and a negative number into a positive number.
- The unary plus (+) does not change the number.
- The decrement operator (--) decrements the value of its operand by 1.
- The increment operator (++) increments the value of its operand by 1.

The prefix version (++x or --x)

- Comes before the operand, as in ++x
- First increments or decrements the variable by 1 and then uses the value of the variable.

```
int x = 5;
int y = ++x;
```

means

Change x
Then assign to y
y = 6, x = 6.

```
int x = 5;
x = x + 1;
int y = x;
```

The postfix version (x++ or x--)

- Comes after the operand, as in x++
- Uses the current value of the variable and then increment or decrements the variable by 1.

```
int z = 5;
int y = z++;
```

means

Assign z to y.
Then change z.
y is 5, z is 6.

```
int z = 5;
int y = z;
z = z + 1;
```

Relational and comparison operators

- The result of such an operation is either true or false (i.e., a Boolean value)

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Example

```
(7 == 5)      // evaluates to false
(5 > 4)       // evaluates to true
(3 != 2)      // evaluates to true
(6 >= 6)      // evaluates to true
(5 < 5)       // evaluates to false
```

```
(a == 5)     // evaluates to false, since a is not equal to 5
(a*b >= c)   // evaluates to true, since (2*3 >= 6) is true
(b+4 > a*c)  // evaluates to false, since (3+4 > 2*6) is false
((b=2) == a) // evaluates to true
```

Logical Operators

- To combine or modify existing expressions.

!	NOT
&&	AND
||	OR

- **Example**

```
a > 5 && b > 5
ch == 'y' || ch == 'Y'
!valid
!(x > 5)
```

Conditional ternary operator (?)

- The conditional operator evaluates an expression, returning one value if that expression evaluates to true, and a different one if the expression evaluates as false.
- Syntax is:

  condition ? result1 : result2

```
7==5 ? 4 : 3
7==5+2 ? 4 : 3
5>3 ? a : b
a>b ? a : b
```

Example

```
// conditional operator
#include <iostream>
using namespace std;

int main ()
{
  int a,b,c;

  a=2;
  b=7;
  c = (a>b) ? a : b;

  cout << c << '\n';
}
```

Comma operator (,)

- The comma operator (,) is used to separate two or more expressions
- has the lowest precedence
- is left-associative
- Example,

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

```
int main()
{
    int a, b;
    a = (b=3, b+2);
    cout << "a is : "<< a;

    return 0;
}
```

```
a is : 5
```

Bitwise operators

(&, |, ^, ~, <<, >>)

Bitwise operators modify variables considering the bit patterns that represent the values they store.

operator	asm equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift bits left
>>	SHR	Shift bits right

Precedence of operators

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right

Precedence of operators

8	Relational	< > <= >=	comparison operators	Left-to-right
9	Equality	== !=	equality / inequality	Left-to-right
10	And	&	bitwise AND	Left-to-right
11	Exclusive or	^	bitwise XOR	Left-to-right
12	Inclusive or	|	bitwise OR	Left-to-right
13	Conjunction	&&	logical AND	Left-to-right
14	Disjunction	||	logical OR	Left-to-right
15	Assignment-level expressions	= *= /= %= += -= >>= <<= &= ^= |=	assignment / compound assignment	Right-to-left
		?:	conditional operator	
16	Sequencing	,	comma separator	Left-to-right

- **Precedence:** when an expression contains two different kinds of operators, which should be applied first?
- **Associativity:** when an expression contains two operators with the same precedence, which should be applied first?

Thank You