

BSc (Hons) in Software Engineering

IT1101: Introduction to Computer
Programming
Lecture 7
Operator Types in C

Topics to be covered

- Operator Types in C
 - Arithmetic Operators
 - Increment and Decrement Operators
 - Logical (or Relational) Operators
 - Bitwise Operators
 - Assignment Operators
 - Misc Operators
- Operator Precedence
- Associativity
- Operator Precedence and Associativity in C

Operator Types in C

- C language supports following types of operators.
 - Arithmetic Operators
 - Logical (or Relational) Operators
 - Bitwise Operators
 - Assignment Operators
 - Misc Operators

Arithmetic Operators

□ Assume `int A = 10 , B = 20;`

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denominator	B / A will give 2

Arithmetic Operators

Operator	Description	Example
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Divide and Modulus Division Operators

- When both operands of the divide operator are integers, the result is the integer quotient. When at least one of the operands is real, the result is the real number quotient.
- To obtain the integer remainder of the division of two integers use the modulus division operator %.
- The modulus division operations should not be used with negative operands since the result is not consistent from one computer to another.

Unary Operators (Needs only one Operand)

- Unary Operators for C are ++ and --.
- There 2 types, i.e. Postfix and Prefix.
- Prefix Unary Eg: A=5; A—
 - Value will change before any inline processing
- Postfix Unary Eg: A=5; A++
 - Value will change after any inline processing

Self Increment and Decrement Operators

- **Prefix (++X)**

- `int a = 10;`
- `printf("a = %d \n", ++a);`
- `printf("a = %d \n", --a);`

- **Postfix (X++)**

- `int a = 10;`
- `printf("a = %d \n", a++);`
- `printf("a = %d \n", a--);`

Sample Code – Prefix & Postfix Operators in use

```
int a=10;
```

```
    a++; printf("After Postfix increment:%d\n",a);
```

```
    printf("Inline Postfix Increment:%d\n",a++); //note that value change after  
display
```

```
    a++;
```

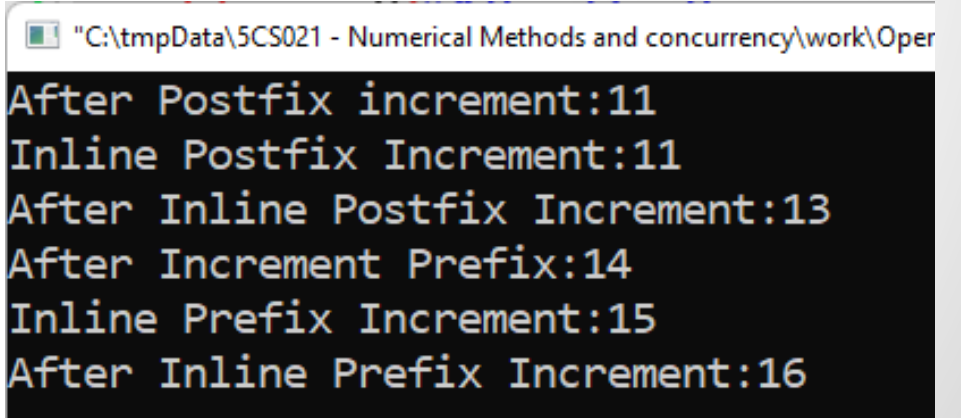
```
    printf("After Inline Postfix Increment:%d\n",a);
```

```
    ++a; printf("After Increment Prefix:%d\n",a);
```

```
    printf("Inline Prefix Increment:%d\n",++a); //note that value change before  
display
```

```
    ++a;
```

```
    printf("After Inline Prefix Increment:%d\n",a);
```



```
"C:\tmpData\5CS021 - Numerical Methods and concurrency\work\Oper
After Postfix increment:11
Inline Postfix Increment:11
After Inline Postfix Increment:13
After Increment Prefix:14
Inline Prefix Increment:15
After Inline Prefix Increment:16
```

Unary Minus

- The unary minus precedes its operand.
- The result of the unary minus is to change the sign of the operand.
- If it was positive the result is negative. If it was negative the result is positive.
- It is the same as multiplying the operand by -1 .

Increment and Decrement using another value - Operators

contd.

- `K=++N; //Prefix increment: N=N+1; then K=N;`
- `K=N++; //Postfix increment: K=N ; then N=N+1;`
- `K=--N; //Prefix decrement: N=N-1; then K=N;`
- `K=N--; //Postfix decrement: K=N ;then N=N-1;`

Exercise – Find the value of result

- `int result, a = 10, b = 4;`
- `result = ++a + b--;` Same as `(++a) + (b--)`
- `result = a++ - --b;`
- `result = b-- - ++a;`
- `result = a++ - b--/2;`

Quick Quiz

- (a) `n = 5;`

`k = n++ + n++;`

What is the final value of `n` and `k`?

- (b) `x = 3;`

`y = --x + --x;`

What is the final value of `x` and `y`?

- (c) `d = 4;`

`e = --d + d++;`

What is the final value of `d` and `e`?

Logical (or Relational) Operators

Operator	Description	Example
==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

Logical (or Relational) Operators

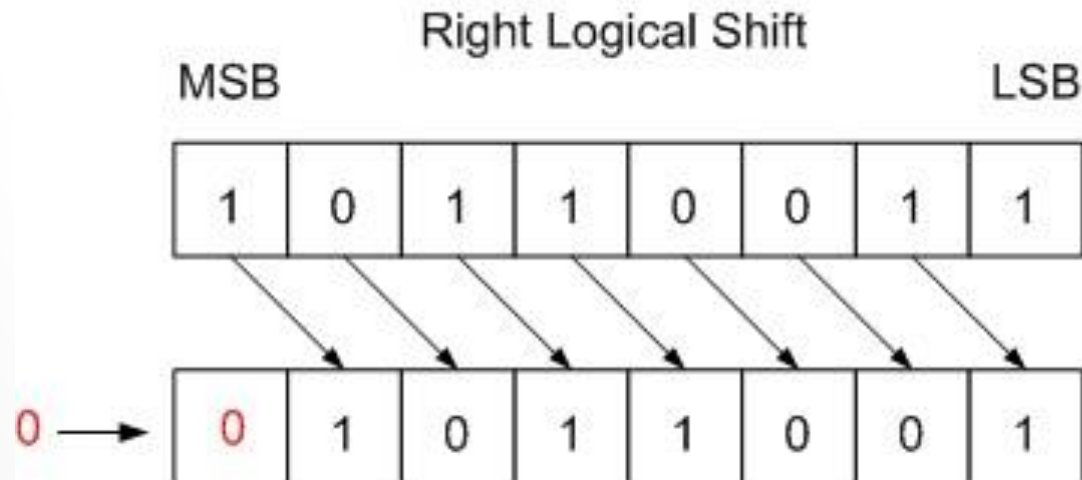
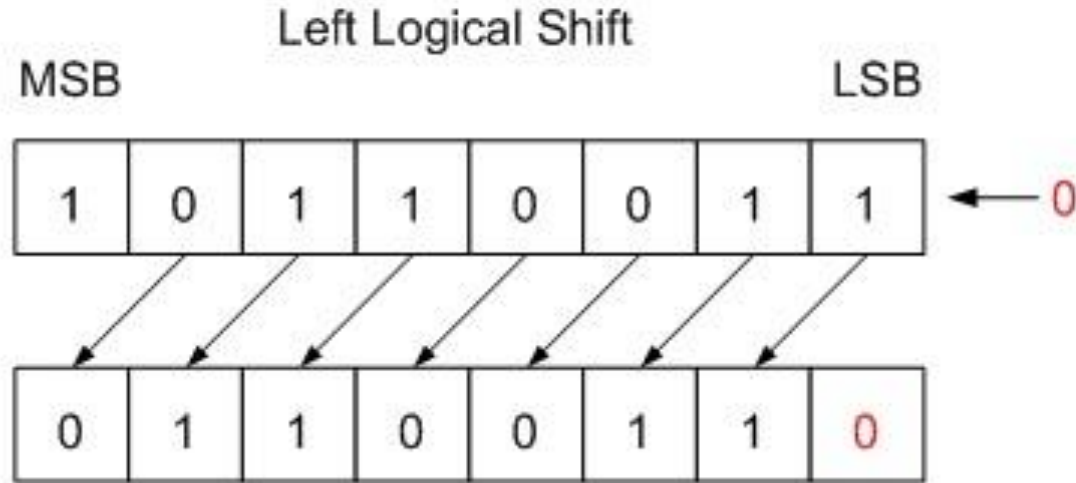
Operator	Description	Example
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is true.
	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Bitwise Operators

- Assume if A = 60; and B = 13;
- Now in binary format they will be as follows:
 - A = 0011 1100
 - B = 0000 1101

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61 which is 0011 1101

Logical shift



Bitwise Operators

Operator	Description	Example
\wedge	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A \wedge B) will give 49 which is 0011 0001
\sim	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(\sim A) will give -60 which is 1100 0011
\ll	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A \ll 2 will give 240 which is 1111 0000
\gg	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A \gg 2 will give 15 which is 0000 1111

Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ will assign value of $A + B$ into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$

Assignment Operators

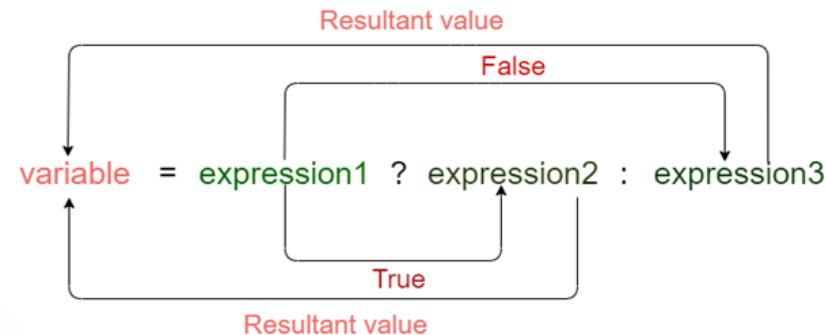
Operator	Description	Example
<code>*=</code>	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	<code>C *= A</code> is equivalent to <code>C = C * A</code>
<code>/=</code>	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	<code>C /= A</code> is equivalent to <code>C = C / A</code>
<code>%=</code>	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	<code>C %= A</code> is equivalent to <code>C = C % A</code>
<code><<=</code>	Left shift AND assignment operator	<code>C <<= 2</code> is same as <code>C = C << 2</code>

Assignment Operators

Operator	Description	Example
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
=	bitwise inclusive OR and assignment operator	C = 2 is same as C = C 2

Misc Operators

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(int), will return 4.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y



Operator Precedence

- Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated.
- Certain operators have higher precedence than others.
- **NOTE : Parentheses can be used to modify the normal order of execution of an expression**

Operator Precedence

- Example: The multiplication operator (*) has higher precedence than the addition operator(+)

➤ Example : $x = 7 + 3 * 2$

✓ Answer = 13, not 20

- If you need the answer 20 you can use ()

➤ Example : $x = (7 + 3) * 2$

✓ Now, answer = 20

Associativity

- The associativity of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses
- Left associative
 - Operands are associated to the operator on their left side,
Left-to-right
- Right associative
 - Operands are associated to the operator on their right side,
Right-to-left

Associatively

- Example:

$$7-4+2$$

- If $-$ and $+$ are Left Associative
 - Answer = 5
 - This is the result you get in C
-
- If $-$ and $+$ are Right Associative
 - Answer = 1

Do it your self...

$$74 / 2 \% 2 * 5 - 10 \% (5 - 1)$$

- First deal with ()
- Next work from left to right on / , % and * operators
- Finally perform the subtraction

Arithmetic Operators and Characters

- C allows the use of the arithmetic operators with characters by using the ASCII representation of the character as if it were an integer.

- Examples

- 'A' + 5 is
- '5' - '0' is
- 'c' - 'a' is

Characters	ASCII Value
A - Z	65 - 90
a - z	97 - 122
0 - 9	48 - 57
Special Symbol	0 - 47, 58 - 64, 91 - 96, 123 - 127

Operator Precedence and Associativity in C

	Category	Operator	Associativity
1	Postfix	() [] -> . ++ - -	Left to right
2	Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
3	Multiplicative	* / %	Left to right
4	Additive	+ -	Left to right
5	Shift	<< >>	Left to right
6	Relational	< <= > >=	Left to right
7	Equality	== !=	Left to right

Operator Precedence and Associativity in C

	Category	Operator	Associativity
8	Bitwise AND	&	Left to right
9	Bitwise XOR	^	Left to right
10	Bitwise OR		Left to right
11	Logical AND	&&	Left to right
12	Logical OR		Left to right
13	Conditional	?:	Right to left
14	Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
15	Comma	,	Left to right

Thank You.