1

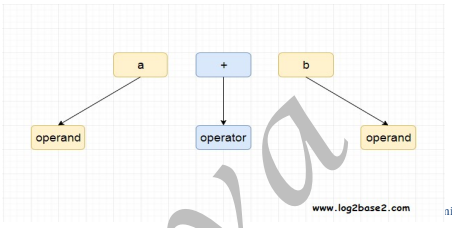
OPERATORS IN C

by
Dr. Sumit Srivastava
Dept. of Computer Science & Engineering

1

Operators in C

- An operator is a symbol that operates on a value or a variable.
- Operator specifies which operation we need to perform.
- Operands are the values that we perform operation on those values.
- Example



2

Operators in C

- Example

z = x – y ;

Here in this example symbol "-" is a subtraction operator that subtracts y from x and stores the evaluated value in the variable z. Here "=" is an assignment operator. x and y are two operands.

- An operator is a symbol that tells the compiler to perform certain mathematical
- or logical manipulations.
- Operators are used in programs to manipulate data and variables.

3

Operators in C

- Precedence:

If there is more than one operator in an expression, which operation we need to perform first is specified by precedence.

- Associativity:

If more than one operator has the same precedence which operation, we need to perform is specified by associativity.

4

Types of operators

Operators in C

	Operators	Type
Unary Operator	++, --	Unery Operator
Binary Operator	+, -, *, /, %	Arithmetic Operator
	<, <=, >, >=, ==, !=	Relational Operator
	&&, ||, !	Logical Operator
	&, |, <<, >>, ~, ^	Bitwise Operator
	=, +=, -=, *=, /=, %=	Assignment Operator
Ternary Operator	?:	Ternary or Conditional Operator

5

Types of operators

- Types of operators based on number of operands:
  1. Unary Operators – require one operand.
  2. Binary Operators- require two operands.
  3. Ternary Operators – require three operands.

6

Unary Operators

7

- Only one operand is required to perform calculation.
  - ++(increment operator)
  - -- (decrement operator)
  - size of operator
- Example
  - a = 5;
  - ++a;
  - --a;

7

Binary Operator

8

- Two operands are required to perform calculation.
  - +, -, *, % etc.
  - logical operator (&&,||,!)
  - Bitwise operator (&,|,<<,>>,~,^)
  - Arithmetic operator (+,-,/,*,%)
  - Relational operator(<,>,>=,<=,==,!=)
  - Assignment operator (=,+=,-=,/=,*=,%=)
- Example
  - a = 5;
  - b = 15;
  - a + b;
  - a - b;

8

Ternary Operator

9

- Three operands are required to perform calculation.
- It is also called as a conditional operator in c.
- Example

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

a > b?printf("a"):printf("b");

9

Types of Operator

10

- Arithmetic Operator
  [+, -, *, /, %]
- Assignment Operator
  [=, +=, -=, *=, /=, %= etc...]
- Increment / Decrement Operator
  [++, --]
- Relational Operator
  [>, <, >=, <=, !=, ==]
- Logical Operator
  [&&, ||, !]
- Bitwise Operator
  [&, |, ^, ~]
- Conditional Operator
  [? :]

10

Arithmetic operator

11

- Arithmetic operators are used to perform basic mathematical calculations.
- Example

Addition, subtraction, multiplication, division, modulo operations.

- The Modulus operator % can only be used with integers.

11

Arithmetic operator

12

Operator	Description	Example: (a=100,b=10,ans=0)
+	Add to operands	**ans = a + b;** ans will be 110.
-	Subtract operand_2 form operand_1	**ans = a - b;** ans will be 90.
*	Multiply two operands	**ans = a * b;** ans will be 1000.
/	divides operand_1 by operand_2	**ans = a / b;** ans will be 10.
%	Gives the remainder after the actual division of operands	**ans = a % b;** ans will be 0. Because 10 will perfectly divide 100 so remainder will be zero. 8 % 3 = 2. 5 % 4 = 1.

12

Increment and Decrement Operators

13

- Using increment and decrement operators, we can increment the value of a variable by 1 or decrement by 1.
- It have two variants-
  - **prefix** (++x or -- x)
  - **postfix** (x++ or x--)

y = x++;	y = ++x;
is equivalent to writing	is equivalent to writing
y = x ;	x = x + 1 ;
x = x + 1;	y = x ;

13

Post-fix notation & Pre-fix notation

14

Post-fix notation

- When post-fix notation is used, the result is evaluated first and then the operation (inc / dec) is performed.

Pre-fix notation

- When pre-fix notation is used, the operation (inc / dec) is performed first then the result is evaluated.

Associativity is from Right -> Left.

14

Increment and Decrement Operators

15

Type of Operator	Sample Operator Expression	Description/ Explanation
Prefix Increment Operator	++p	p increases by a value of 1, then the program uses the value of p.
Postfix Increment Operator	p++	The program uses the current value of p and increases the value of p by 1.
Prefix Decrement Operator	–p	p decreases by a value of 1, then the program uses the value of p.
Postfix Decrement Operator	p–	The program uses the current value of p and decreases the value of p.

15

Post-fix & Pre-fix Example:

16

```
// x initialized to 5
int x = 5;
int y = x++; (postfix)
//y assigned the value of x
//x is incremented
[RESULT : x = 6, y = 5]
```

```
// x initialized to 5
int x = 5;
int z = ++x; (prefix)
//x is incremented
//a assigned the value of x
[RESULT : x = 6, z = 6]
```


16

Pre-Increment & Post-increment

17

pre-increment

- In the case of pre-increment, the value of the variable is increased by one before the expression evaluation.
- It means in pre-increment, first the value of the variable is incremented by one, then the modified value is used in the expression evaluation.

post-increment

- In the case of post-increment, the value of the variable is increased by one after the expression evaluation.
- It means, in post-increment, first the expression is evaluated with existing value, then the value of the variable is incremented by one.

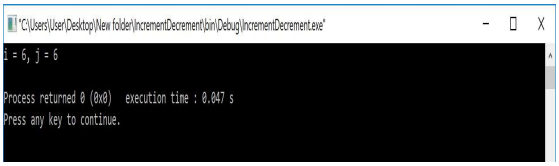
17

Pre-Increment

18

Example Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int i = 5,j;
  j = ++i; // Pre-Increment
  printf("i = %d, j = %d",i,j);
}
```



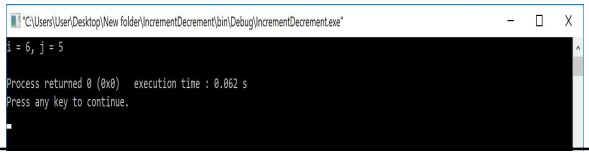
18

Post-Increment

19

- Example Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i = 5,j;
   j = i++;  // Post-Increment
   printf("i = %d, j = %d",i,j);
}
```



19

Relational Operators

20

- A relational operator checks the relationship between two operands.
- If the relation is true, it returns 1.
- If the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.

20

Relational Operators

21

- <, <=, >, >= ---------- highest precedence
- ==,!= ---------- lowest precedence
- Arithmetic operators have a higher priority than relational operators.
- Associativity is from Left to Right.

ae-1 relational operator ae-2

Where ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables, or combination of them.

21

Relational Operators

22

Greater Than: >
int a = 5, b = 10;
(a > b)
[Result: 0]

2. Less Than: <
int a = 5, b = 10;
(a < b)
[Result: 1]

3. Equal To: ==
int a = 5, b = 5;
(a == b)
[Result: 1]

4. Not Equal To: !=
int a = 5, b = 2
(a != b)
[Result: 1]

5.Greater Than Equal: >=
int a = 10, b = 5;
(a >= b)
[Result: 1]

6.Less Than Equal: <=
int a = 10, b = 5;
(a <= b)
[Result: 0]

22

Relational Operators

23

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

23

Assignment Operator

24

- Assignment operator can be used to assign the value to a variable.
- The assignment operator has the lowest precedence of all operator.
- It is always evaluated last.
- In assignment, the previous value of the variable is overwritten by the value of the expression.

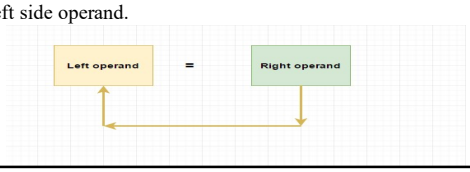
variable = expression

24

Assignment Operator

25

- Example:
  a=10;
- Left hand side must be a variable; Right hand side may be a variable/constant or combination of these two.
- Basically, the value of right-side operand will be assigned to the left side operand.



25

Assignment Operator

26

- 'C' has a set of 'shorthand' assignment operators of the form

  v op=exp;

  where v is a variable, exp is an expression and op is C binary arithmetic operator.
- The operator **op**=is known as the shorthand assignment operator.
- Associativity is from Right to Left.
- The assignment statement

  vop= exp;

  is equivalent to

  v = v op (exp);

26

27

Statement with simple assignment operator	Statement with shorthand operator
a =a + 100	a += 100
a = a - 20	a -= 20
a = a * (n+1)	a *= (n+1)
a = a / (n+1)	a /= (n+1)
a = a % b	a %= b

27

Assignment Operator

28

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15
+=	Add both left and right-hand side values and store the result into left-hand side variable	A += 10 ⇒ A = A+10
-=	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	A -= B ⇒ A = A-B
*=	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	A *= B ⇒ A = A*B
/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	A /= B ⇒ A = A/B
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	A %= B ⇒ A = A%B

28

Compound assignment operators

29

Operator	Meaning	Example (a = 10 , b = 5)
+=	L=L+R add left and right operand and assign result in left	**a+=b;**same as a=a+b after execution **a** will hold 15
-=	L=L-R subtract right operand from left operand and assign result in left	**a-=b;**same as a=a-b after execution **a** will hold 5
*=	L=L*R multiply both right and left operand and store result in left	**a*=b;**same as a=a*b after execution **a** will hold 50
/=	L=L/R divides left operand by right operand and store result in left	**a/=b;**same as a=a/b after execution **a** will hold 2
%=	L=L%R After left and right operand division, the remainder will be stored in left	**a%=b;**same as a=a%b after execution **a** will hold 0

29

Assignment Operator

30

Add Assignment +=

```
int a = 20;
//add 5 to a
a += 5;  similar to a = a+5;
printf("%d",a);
```

[OUTPUT : 25]

2. Multiply Assignment *=

```
int a = 20;
//multiply 5 to a
a *= 5;  similar to a = a*5;
printf("%d",a);
```

[OUTPUT : 100]

Similarly, we have -=, :/=, %=, <<=, >>=, &=, ^=, |=

30

Logical Operators

31

- Logical Expression
  - An expression whose value is a boolean type (true (1) or false (0)).
- Logical operators are used when we want to compare more than one relation at a time.
- There are three logical operators
  1. || (Logical OR) operator
  2. && (Logical AND) operator
  3. ! (Logical NOT) operator

31

|| (Logical OR) operator

32

- If one of the operands or expressions is true, it will return 1.
- If all of them are false, it will return 0.

A	B	A || B	Example
0	0	0	(5 > 10) || (5 < 4) Both expressions are false. so, logical OR output will be 0
0	1	1	(10 > 20) || (10 < 20) First expression is false and second one is true. so, logical OR output will be 1
1	0	1	(10 < 20) || (10 > 100) First expression is true and second one is false. so, logical OR output will be 1
1	1	1	(10 < 20) || (10 < 100) Both expressions are true. so, logical OR output will be 1

32

&& (Logical AND) Operator

33

- If both left and right operands or expressions are true, it will return true. Otherwise, it will return false.
- Note, non-zero value operands are considered as true.

A	B	A && B	Example
0	0	0	(5 > 10) && (5 < 4) Both expressions are false. so, logical AND output will be 0
0	1	0	(10 > 20) && (10 < 20) First expression is false and second one is true. so, logical AND output will be 0
1	0	0	(10 < 20) && (10 > 100) First expression is true and second one is false. so, logical AND output will be 0
1	1	1	(10 < 20) && (10 < 100) Both expressions are true. so, logical AND output will be 1

33

! (Logical NOT) Operator

34

- Logical NOT operator is used to inverse the current decision.

Say, if current state is true, Logical NOT (!) operator will make it false.

A	!A	Example
0	1	!(100 < 10) 100 is greater than 10. So, it will return false. !(false) ==> true
1	0	!(10 < 100) 10 is less than 100. So, it will return true. !(true) ==> false

34

Logical Operators: Example

35

1. Logical AND (&&)

```
int x = 6;
int b = (x > 5)&&(x < 10);
printf("%d", b );
```

[OUTPUT: 1]

```
int x = 6;
int b = (x > 5)&&(x !=6);
printf("%d", b );
```

[OUTPUT: 0]

2. Logical OR (||)

```
int x = 6;
int b = (x > 5)||(x !=6);
printf("%d", b );
```

[OUTPUT: 1]

---

3. Logical NOT (!)

```
int x = 6;
int b = !(x > 5)
printf("%d", b );
```

[OUTPUT: 0]

35

Logical Operators

36

- Local NOT ---------- highest precedence,

  Associativity from Right-> Left
- Logical AND ---------- next highest precedence,

  Associativity from Left -> Right
- Logical OR ---------- lowest precedence,

  Associativity from Left -> Right

36

Bitwise Operators

37

- Bitwise operators performs operations on bits.
- Bitwise operators may not be applied to float or double.
- AND (&), OR (|), XOR (^), NOT (~).
- **Truth Table:-**

A	B	A&B	A|B	A^B	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

37

Bitwise Operators; Example

38

```
#include<stdio.h>
int main( ) {
int a = 6;
int b = 3;
int r1 = a & b;
int r2 = a | b;
int r3 = a ^ b;
int r4 = ~a;
printf("%d, %d, %d, %d",r1,r2,r3,r4);
return 0; }
```

[**OUTPUT:** 2, 7, 5, -7]

DESCRIPTION

```
a    = 0000 0110
b    = 0000 0011
--------------------
a&b = 0000 0010 = 2
--------------------
a|b =  0000 0111 = 7
--------------------
a^b =  0000 0101 = 5
--------------------
~a =  1111 1001 = -7
--------------------
```

38

Bitwise Operators; Example

39

~a = 1111 1001 = -7

In ~a the first bit(Most significant bit) is 1 that represents number is negative, so for negative values system evaluates 2s compliment of the bit and print magnitude with minus (-) symbol.

So,

~6 = 1111 1001

1s 0f ~6 = 0000 0110

+1

2s of ~6 = 0000 0111 = 7

---

so, result is -7.

39

Bitwise Operators

40

Operator	Also known as
<<	Binary Left Shift Operator
>>	Binary Right Shift Operator
~	Binary Ones Complement Operator
&	Binary AND Operator
^	Binary XOR Operator
|	Binary OR Operator

40

Bitwise Operators

41

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
|	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A | B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A ⇒ 6 (00110)
<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 ⇒ 6 (00110)

41

Conditional Operator (? :)

42

- Conditional operator is also known as the ternary operator.
- It takes three operands and evaluates the boolean expression.

42

Conditional Operator (? :)Example

43

- Conditional operator is also known as the ternary operator.
- It takes three operands and evaluates the boolean expression.

```
#include<stdio.h>
 int main()
 {
int a = 10, b = 20, max;
 max = ( a > b) ? a : b;
printf("Max = %d",max);
 return 0;
 }
```

- Output: **Max=20**

43

Other Operators

44

- Comma Operator
- The sizeof operator ('sizeof' is a unary operator that returns the size of data (constants, variables, array, structure, etc).
- ternary operator (?☺
- reference operator (&)
- dereference operator (*)
- member selection operator (->)

44

45

1. sizeof():- If you want to check the size of data types available in C then you can do it by using sizeof() operator.

2. Reference Operator (&):– Used for returning the address of a memory location.

3. Pointer Operator (*):– It is a pointer to a variable.

45

Other Operators

46

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

46

Other Operators (Example)

47

```
#include <stdio.h>
 main()
{
 int a = 4;
short b;
double c;
 int* ptr;

/* example of sizeof operator */
 printf("Line 1 - Size of variable a = %d\n", sizeof(a) );
 printf("Line 2 - Size of variable b = %d\n", sizeof(b) );
 printf("Line 3 - Size of variable c= %d\n", sizeof(c) );

/* example of & and * operators */
ptr = &a;
 /* 'ptr' now contains the address of 'a'*/
 printf("value of a is %d\n", a);
printf("*ptr is %d.\n", *ptr);

/* example of ternary operator */
 a = 10; b = (a == 1) ? 20: 30;
printf( "Value of b is %d\n", b );

b = (a == 10) ? 20: 30;
printf( "Value of b is %d\n", b );
 }
```

```
Line 1 - Size of variable a = 4
Line 2 - Size of variable b = 2
 Line 3 - Size of variable c= 8
value of a is 4 *ptr is 4.
 Value of b is 30 Value of b is 20
```

47

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

48

Arithmetic expressions

49

- a + b * c – d / e ➔ a + (b * c) – (d / e)
- a * -b + d % e – f ➔ a * (-b) + (d % e) – f
- a – b + c + d ➔ (((a – b) + c) + d)
- x * y * z ➔ ((x * y) * z)
- a + b + c * d * e ➔ (a + b) + ((c * d) * e)

49

50

THANK YOU

50