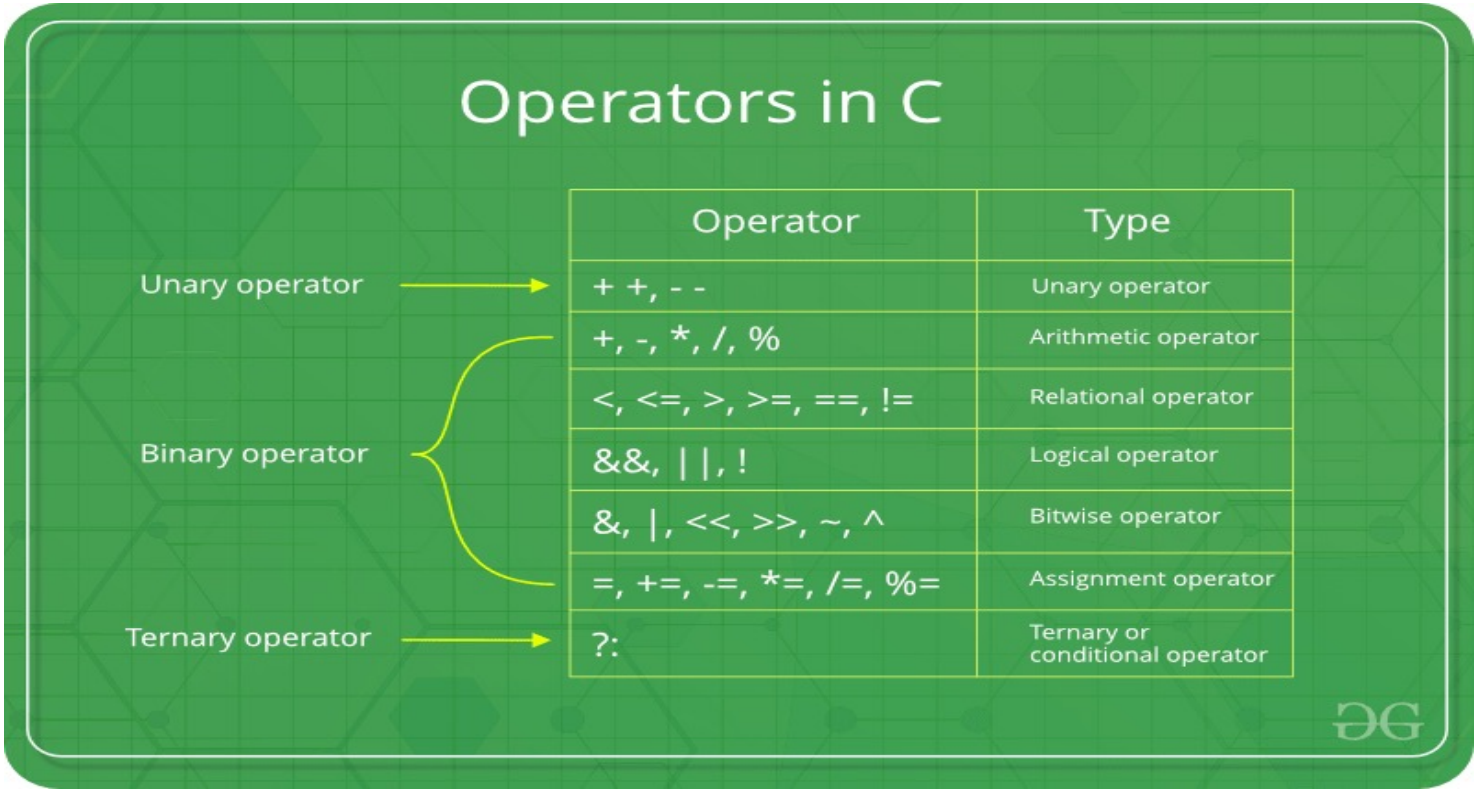
Operators in C

Lesson Objectives

- *Arithmetic operators*
- *Relational operators*
- *Logical operators*
- *Bitwise operators*
- *Assignment operators*
- *Conditional operators*
- *Special operators*

Operators in C



Section 1

ARITHMETIC OPERATORS

Arithmetic operators (1)

C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

Arithmetic operators (2)

These are the operators used to perform arithmetic/mathematical operations on operands. Examples: (+, -, *, /, %,++,–). Arithmetic operator are of two types:

* Unary Operators: Operators that operates or works with a single operand are unary operators. For example: (++ , –)

* Binary Operators: Operators that operates or works with two operands are binary operators. For example: (+ , – , * , /)

Arithmetic operators (3)

```c
#include <stdio.h>
int main()
{
   int a = 10, b = 4, res;
   printf("a is %d and b is %d\n", a, b); // printing a and b
   res = a + b; // addition
   printf("a+b is %d\n", res);
   res = a - b; // subtraction
   printf("a-b is %d\n", res);
   res = a * b; // multiplication
   printf("a*b is %d\n", res);
   res = a / b; // division
   printf("a/b is %d\n", res);
   res = a % b; // modulus
   printf("a%b is %d\n", res);
   return 0;
}
```

Output:

```
a is 10 and b is: 4
a+b is: 14
a-b is: 6
a*b is: 40
a/b is: 2
a%b is: 2
```

Arithmetic operators (4)

```c
#include <stdio.h>
int main()
{
   int a = 10, b = 4, res;
   // post-increment example: res is assigned 10 only, a is not updated yet
   res = a++;
   printf("a is %d and res is %d\n", a, res); // a becomes 11 now
   // post-decrement example: res is assigned 11 only, a is not updated yet
   res = a--;
   printf("a is %d and res is %d\n", a, res); // a becomes 10 now
   // pre-increment example: res is assigned 11 now since a is updated here itself
   res = ++a;
   // a and res have same values = 11
   printf("a is %d and res is %d\n", a, res);
   // pre-decrement example: res is assigned 10 only since a is updated here itself
   res = --a;
   // a and res have same values = 10
   printf("a is %d and res is %d\n", a, res);
   return 0;
}
```

Output:

```
a is 11 and res is 10
a is 10 and res is 11
a is 11 and res is 11
a is 10 and res is 10
```

Section 2

RELATIONAL OPERATORS

Relational operators (1)

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

Relational operators (2)

```c
#include <stdio.h>
int main()
{
  int a = 21, b = 10, c;
  if( a == b ) { printf("Line 1 - a is equal to b\n" );}
   else { printf("Line 1 - a is not equal to b\n" );}
  if ( a < b ) { printf("Line 2 - a is less than b\n" );}
   else { printf("Line 2 - a is not less than b\n" );}
  if ( a > b ) { printf("Line 3 - a is greater than b\n" );}
   else { printf("Line 3 - a is not greater than b\n" );}
  /* Lets change value of a and b */
  a = 5;   b = 20;
  if ( a <= b ) {
printf("Line 4 - a is either less than or equal to  b\n" );}
  if ( b >= a ) {
printf("Line 5 - b is either greater than  or equal to b\n" );}
  return 0;
}
```

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to  b
Line 5 - b is either greater than  or equal to b
```

Section 3

LOGICAL OPERATORS

Logical Operators (1)

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Logical Operators (2)

```c
#include <stdio.h>
int main()
{
   int a = 5, b = 20, c ;
   if ( a && b ) { printf("Line 1 - Condition is true\n" );}
   if ( a || b ) { printf("Line 2 - Condition is true\n" );}
   /* lets change the value of  a and b */
   a = 0;   b = 10;
   if ( a && b ) { printf("Line 3 - Condition is true\n" );}
   else { printf("Line 3 - Condition is not true\n" );}
   if ( !(a && b) ) { printf("Line 4 - Condition is true\n" );}
   return 0;
}
```

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

Section 4

BITWISE OPERATORS

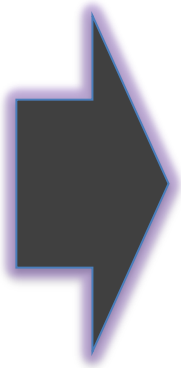
Bitwise Operators (1)

Assume A = 60 and B = 13 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) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Bitwise Operators (2)

```c
#include <stdio.h>
int main()
{
   // a = 5(00000101), b = 9(00001001)
   unsigned char a = 5, b = 9;
   printf("a = %d, b = %d\n", a, b);
   printf("a&b = %d\n", a & b);    // The result is 00000001
   printf("a|b = %d\n", a | b);    // The result is 00001101
   printf("a^b = %d\n", a ^ b);    // The result is 00001100
   printf("~a = %d\n", a = ~a);    // The result is 11111010
   printf("b<<1 = %d\n", b << 1);  // The result is 00010010
   printf("b>>1 = %d\n", b >> 1);  // The result is 00000100
   return 0;
}
```



Output:

```
a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4
```

Section 5

ASSIGNMENT OPERATORS

Assignment Operators (1)

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Assignment Operators (2)

```c
#include <stdio.h>
int main() {
  int a = 21, c;
  c =  a; printf("Line 1 - =  Operator Example, Value of c = %d\n", c );
  c +=  a; printf("Line 2 - += Operator Example, Value of c = %d\n", c );
  c -=  a; printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
  c *=  a; printf("Line 4 - *= Operator Example, Value of c = %d\n", c );
  c /=  a; printf("Line 5 - /= Operator Example, Value of c = %d\n", c );
  c  = 200;
  c %=  a; printf("Line 6 - %= Operator Example, Value of c = %d\n", c );
  c <<=  2; printf("Line 7 - <<= Operator Example, Value of c = %d\n", c );
  c >>=  2; printf("Line 8 - >>= Operator Example, Value of c = %d\n", c );
  c &=  2; printf("Line 9 - &= Operator Example, Value of c = %d\n", c );
  c ^=  2; printf("Line 10 - ^= Operator Example, Value of c = %d\n", c );
  c |=  2; printf("Line 11 - |= Operator Example, Value of c = %d\n", c );
  return 0;
}
```

OUTPUT:

```
Line 1 - =  Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - %= Operator Example, Value of c = 11
Line 7 - <<= Operator Example, Value of c = 44
Line 8 - >>= Operator Example, Value of c = 11
Line 9 - &= Operator Example, Value of c = 2
Line 10 - ^= Operator Example, Value of c = 0
Line 11 - |= Operator Example, Value of c = 2
```

Section 6

CONDITIONAL OPERATOR

Conditional operator (1)

The conditional operators in C language are known by two more names

+) **Ternary Operator**

+) **? : Operator**

It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is :

```
expression 1 ? expression 2: expression 3
```

Conditional operator (2)

```
expression 1 ? expression 2: expression 3
```

The question mark "?" in the syntax represents the if part.
The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
If (expression 1) returns true then the expression on the left side of " : " i.e (expression 2) is executed.
If (expression 1) returns false then the expression on the right side of " : " i.e (expression 3) is executed.

Section 7

SPECIAL OPERATORS

Special Operators (1)

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x ;** return address of the variable **x**
*	Pointer to a variable	***x ;** will be pointer to a variable **x**

sizeof operator: sizeof is a much used in the C/C++ programming language. It is a compile time unary operator which can be used to compute the size of its operand. The result of sizeof is of unsigned integral type which is usually denoted by size_t. Basically, sizeof operator is used to compute the size of the variable. To learn about sizeof operator in details you may visit this link.

Special Operators (2)

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x ;** return address of the variable **x**
*	Pointer to a variable	***x ;** will be pointer to a variable **x**

Comma Operator: The comma operator (represented by the token ,) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type). The comma operator has the lowest precedence of any C operator. Comma acts as both operator and separator. To learn about comma in details visit this link.

Special Operators (3)

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x ;** return address of the variable **x**
*	Pointer to a variable	***x ;** will be pointer to a variable **x**

Conditional Operator: Conditional operator is of the form Expression1 ? Expression2 : Expression3 .Expression1 is the condition to be evaluated. If the condition(Expression1) is True then we will execute and return the result of Expression2 otherwise if the condition(Expression1) is false then we will execute and return the result of Expression3. We may replace the use of if..else statements by conditional operators.

Special Operator: Ternary Operator

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

int main()
{
  int test = 0; s
  cout << "First  character " << '1' << endl;
  cout << "Second character " << (test ? 3 : '1')
<< endl;

  return 0;
}
```

```
First  character 1
Second character 49
```

Operator precedence (1)

Precedence of operator decreases from top to bottom.

OPERATOR	DESCRIPTION	ASSOCIATIVITY
()	Parentheses (function call)	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++/–	Postfix increment/decrement	
++/–	Prefix increment/decrement	right-to-left
+/-	Unary plus/minus	

Operator precedence (2)

!~	Logical negation/bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
*,/,%	Multiplication/division/modulus	left-to-right
+/-	Addition/subtraction	left-to-right
<< , >>	Bitwise shift left, Bitwise shift right	left-to-right

Operator precedence (3)

< , <=	Relational less than/less than or equal to	left-to-right
> , >=	Relational greater than/greater than or equal to	left-to-right
== , !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
?:	Ternary conditional	right-to-left

Operator precedence (4)

=	Assignment	right-to-left
+= , -=	Addition/subtraction assignment	
*= , /=	Multiplication/division assignment	
%= , &=	Modulus/bitwise AND assignment	
^= , |=	Bitwise exclusive/inclusive OR assignment	
<>=	Bitwise shift left/right assignment	
,	expression separator	left-to-right

Thank you

Q&A