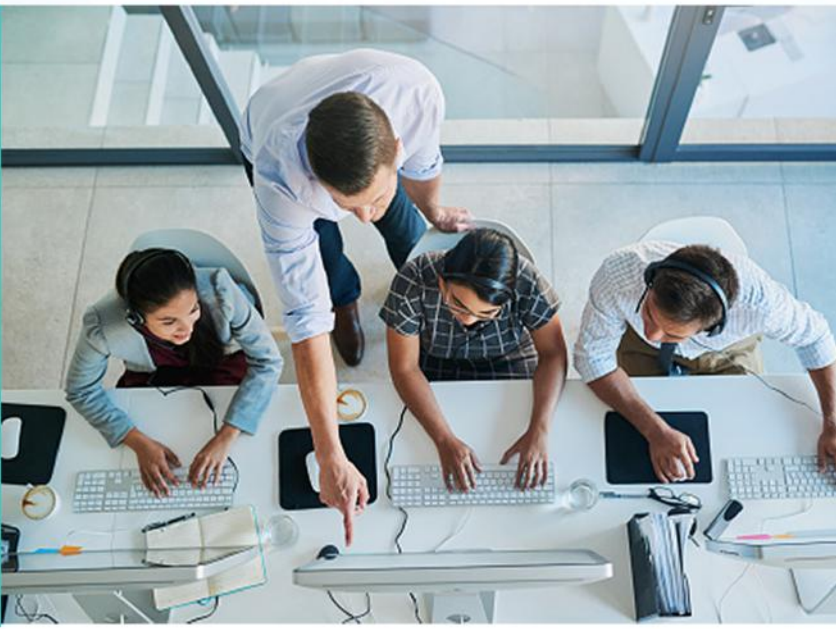
Andhra Pradesh State Skill Development Corporation

Skill AP
APSSDC

eSkill AP
Learn Anytime Anywhere
Andhra Pradesh State Skill Development Corporation



Programming in C

Operators

Operators in C

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators:

Following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B =-10
*	Multiplies both operands.	A * B =200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Example of arithmetic operators:

```
#include <stdio.h>
```

```
main() {

   int a = 21;
   int b = 10;
   int c ;

   c = a + b;
   printf("Line 1 - Value of c is %d\n", c );

   c = a - b;
   printf("Line 2 - Value of c is %d\n", c );

   c = a * b;
   printf("Line 3 - Value of c is %d\n", c );

   c = a / b;
   printf("Line 4 - Value of c is %d\n", c );

   c = a % b;
   printf("Line 5 - Value of c is %d\n", c );

   c = a++;
   printf("Line 6 - Value of c is %d\n", c );

   c = a--;
   printf("Line 7 - Value of c is %d\n", c );
}
```

Output:

```
        Line 1 - Value of c is 31
        Line 2 - Value of c is 11
        Line 3 - Value of c is 210
        Line 4 - Value of c is 2
        Line 5 - Value of c is 1
        Line 6 - Value of c is 21
        Line 7 - Value of c is 22
```

Relational Operators:

Following table shows all the relational operators supported by the C language. Assume variable A holds 10 and variable B holds 20, then:

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.

Examples for relational operators:

```
#include <stdio.h>
main() {

  int a = 21;
  int b = 10;
  int 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" );
  }
}
```

Output:

```
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
```

Logical Operators:

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then:

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 the Logical NOT operator will make it false.	!(A && B) is true.

Example:

```
#include <stdio.h>

main() {
  int a = 5;
  int b = 20;
  int c ;

  if ( a && b ) {
    printf("Line 1 - Condition is true\n" );
  }

  if ( a || b ) {
    printf("Line 2 - Condition is true\n" );
  }

  /* let's 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" );
  }

}
```

Output:

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

Bitwise Operators:

Bitwise operators work on bits and perform bit-by-bit operations. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Assume if A = 60; and B = 13; now in binary format they will be as follows:

```
A = 0011 1100
B = 0000 1101
-------------------------
A&B = 0000 1100

A|B = 0011 1101
A^B = 0011 0001
~A = 1100 0011
```

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then:

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,. 1100 0011
<<	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

Example:

```
#include <stdio.h>

main() {

   unsigned int a = 60; /* 60 = 0011 1100 */
   unsigned int b = 13; /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;       /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;       /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;       /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );

   c = ~a;          /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );

   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

   c = a >> 2;     /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```

Output:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

Assignment Operators:

There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	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

Example:

```
#include <stdio.h>

main() {

  int a = 21;
  int 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 );
}
```

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
```

Misc Operators ↦ sizeof & ternary :

There are few other important operators including sizeof and ? : supported by C Language.

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

Example:

```
#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 );
}
```

Output:

```
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
```