RV College of Engineering®

Go, change the world

UNIT-I
Operators

Introduction

- ***Arithmetic operators***
- ***Relational operators***
- ***Logical Operators***
- ***Assignment operators***
- ***Increment and decrement operators***
- ***Conditional operators***
- ***Bit-wise operators***
- ***Special operators.***

Introduction

- An operator is a symbol, which helps the user to command the computer to do a certain mathematical or logical manipulations.

- Operators are used in programming language to operate on data and variables.

Arithmetic Operators

- C provides five binary ***arithmetic operators:***

  +addition

  - subtraction

  * multiplication

  / division

  % remainder

- An operator is ***binary if it has two operands.***
- There are also two ***unary arithmetic operators:***

  +unary plus

  - unary minus

Unary and Binary Arithmetic Operators

- The unary operators require one operand:

```
i = +1;

j = -i;
```

- The unary + operator does nothing. It's used primarily to emphasize that a numeric constant is positive.

- The value of i % j is the remainder when i is divided by j. 10 % 3 has the value 1, and 12 % 4 has the value 0.

- Binary arithmetic operators—with the exception of %—allow either integer or floating-point operands, with mixing allowed.
- When int and float operands are mixed, the result has type float.

9 + 2.5f has the value 11.5, and 6.7f / 2 has the value 3.35.

The / and % Operators

The / and % operators require special care:

- When both operands are integers, / "truncates" the result. The value of 1 / 2 is 0, not 0.5.
- The % operator requires integer operands; if either operand is not an integer, the program won't compile.
- Using zero as the right operand of either / or % causes undefined behavior.
- The behavior when / and % are used with negative operands is ***implementation-defined in C89.***
- In C99, the result of a division is always truncated toward zero and the value of i % j has the same sign as i.

Operator precedence

- Does i+j*k mean " add i and j,then multiply the result by k" or "multiply j and k, then add i"?
- One solution to this problem is to add parentheses, writing either (i + j) * k or i + (j * k).
- If the parentheses are omitted, C uses ***operator precedence rules to determine the meaning of the*** expression.

- The arithmetic operators have the following relative precedence:

  Highest: + -(unary)
  * / %
  Lowest: + -(binary)

- Examples:

  i + j * k is equivalent to i + (j * k)
  -i * -j is equivalent to (-i) * (-j)
  +i + j / k is equivalent to (+i) + (j / k)

Operator and Associativity

- Associativity Comes into play when an expression contain two or more operators with equal precedence.
  - An operator is said to be ***left associative if it*** groups from left to right.
  - The binary arithmetic operators (*, /, %, +, and -) are all left associative, so

    i - j – k is equivalent to (i - j) - k

    i * j / k is equivalent to (i * j) / k
  - An operator is ***right associative if it groups from*** right to left.
  - The unary arithmetic operators (+ and -) are both right associative, so

    - + i is equivalent to -(+i)

Relational Operators

Relational expressions evaluate to the integer values 1 (true) or 0 (false).

All of these operators are called **binary operators** because they take two expressions as
operands.

< less than
> greater than
<= less than or equal to
>= greater than or equal to
== is equal to
!=is not equal to

Example: int a = 1, b = 2, c = 3 ;

Expression	Value	Expression	Value
a < c	T	a + b >= c	T
b <= c	T	a + b == c	T
c <= a	F	a != b	T
a > b	F	a + b != c	F
b >= c	F		

Logical Operators

An operator that compare or evaluate logical and relationalexpressions.

The following are logicaloperators:

&& LogicalAND

|| LogicalOR

! Logical NOT

Logical Operators

Logical AND

This operator is used to evaluate two conditions or expressions with relational operators simultaneously.

If both the expressions to the left and to the right of the logical operator is true then the whole compound expression is true.

Exp1	Exp2	Exp1 && Exp2
False	False	False
True	False	False
False	True	False
True	True	True

Example:

(a > b) && (x == 10)

The expression to the left is a>b and that on the right is x==10, the whole expression is true only if both expressions are true i.e., if a is greater than b and x is equal to 10.

Logical Operators

Logical AND

Example:

Given a=2, b=3 and c=5,evaluate the following logical expressions:

i. (a > b) && (c != 5)= False

ii. (a < b) && (c < b)= False

iii. (a > b) && (c == 5) = False

iv. (a < b) && (b < c)= True

Logical Operators

Logical OR

- Thelogical OR is used to combine two expressions or the condition evaluates to true if any one of the 2 expressions is true.
- The expression evaluates to true if any one of them is true or if both of them are true.

Exp1	Exp2	Exp1 || Exp2
False	False	False
True	False	True
False	True	True
True	True	True

Example:

(a < m) || (a < n)

The expression evaluates to true if any one of them is true or if both of them are true.

Logical Operators

Logical OR

Example:

Given a=2, b=3 and c=5,evaluate the following logical expressions:

i. (a > b) || (c != 5)= False

ii. (a < b) || (c < b)= True

iii. (a > b) || (c == 5) = True

iv. (a < b) || (b < c)= True

Logical Operators

Logical NOT

- The logical NOT operator takes single expression and evaluates to true if the expression is false

and evaluates to false if the expression is true.

- In other words it just reverses the value of the expression.

Exp1	**!Exp1**
True	False
False	True

Example:

```
! (x >= y)
```

The NOT expression evaluates to true only if the value of x is neither greater than or equal to

Logical Operators

Logical NOT

Example:

Given a=2, b=3and c=5, evaluate the following logical expressions:

i) !(a > b)= True

ii) !(a < b) = False

iii) !(a > b || c == 5)= False

Assignment Operator

- ***Simple assignment: used for storing a value into a*** variable
- ***Compound assignment: used for updating a value*** already stored in a variable

Simple Assignment

- The effect of the assignment *v = e is to evaluate* the expression *e and copy its value into v.*
- *e can be a constant, a variable,* or more complicated expression:

```
i = 5;                  /* i is now 5 */
j = i;                  /* j is now 5 */
k = 10 * i + j;         /* k is now 55 */
```

- If *v and e don't have the same type, then the value* of *e is converted to the type of v as the assignment* takes place:

```
int i;
float f;
i = 72.99f;     /* i is now 72 */
f = 136;        /* f is now 136.0 */
```

Assignment Operator

Simple Assignment

- In many programming languages, assignment is a statement; in C, however, assignment is an operator, just like +.

  The value of an assignment *v = e is the value of v* after the assignment.

  The value of i = 72.99f is 72 (not 72.99).

Side Effects

- An operators that modifies one of its operands is said to have a ***side effect.***
- The simple assignment operator has a side effect: it modifies its left operand.
- Evaluating the expression i = 0 produces the result 0 and—as a side effect—assigns 0 to i.

Since assignment is an operator, several assignments can be chained together:

```
i = j = k = 0;
```

- The = operator is right associative, so this assignment is equivalent to

```
i = (j = (k = 0));
```

Assignment Operator

Side Effects

- Watch out for unexpected results in chained assignments as a result of type conversion:

```
int i;
    float f;
        f = i = 33.3f;    //  i is assigned the value 33, then f is assigned 33.0 (not 33.3).
```

An assignment of the form *v = e is allowed* wherever a value of type *v would be permitted:*

```
i = 1;
k = 1 + (j = i);
printf("%d %d %d\n", i, j, k); /* prints "1 1 2" */
```

- "Embedded assignments" can make programs hard to read.
- They can also be a source of subtle bugs.

Assignment Operator

Lvalues

- The assignment operator requires an ***lvalue as its*** left operand.
- An lvalue represents an object stored in computer memory, not a constant or the result of a computation.
- Variables are lvalues; expressions such as 10 or 2 * i are not.

Since the assignment operator requires an lvalue as its left operand, it's illegal to put any other kind of expression on the left side of an assignment expression:

```
12 = i; /*** WRONG ***/
i + j = 0; /*** WRONG ***/
-i = j; /*** WRONG ***/
```

- The compiler will produce an error message such as *"invalid lvalue in assignment."*

Assignment Operator

Compound Assignment

- Assignments that use the old value of a variable to compute its new value are common.
- Example:

```
i = i + 2;
```

- Using the += compound assignment operator, we simply write:

```
i += 2;   /* same as i = i + 2; */
```

There are nine other compound assignment operators, including the following:

```
-= *= /= %=
```

- All compound assignment operators work in much the same way:

v += e adds v to e, storing the result in v
v -= e subtracts e from v, storing the result in v
*v *= e multiplies v by e, storing the result in*

v v /= e divides v by e, storing the result in v

Assignment Operator

v += e isn't "equivalent" to v = v + e.

- One problem is operator precedence: i *= j + k isn't the same as i = i * j + k.
- There are also rare cases in which *v += e differs* from *v = v + e because v itself has a side effect.*
- Similar remarks apply to the other compound assignment operators.

When using the compound assignment operators, be careful not to switch the two characters that make up the operator.

- Although i =+ j will compile, it is equivalent to i = (+j), which merely copies the value of j into i.