Operators and Expressions

Contents

3.1 Introduction

- An operator is a symbol that tells the computer to perform certain manipulations.
- An expression is a sequence of operands and operators that reduces to a single value.
- C operators can be classified into a number of categories.
  - Arithmetic operators
  - Relational operators
  - Logical operators
  - Assignment operators
  - Increment and decrement operators
  - Conditional operators
  - Bitwise operators
  - Special operators

3.2 Arithmetic operators

- The arithmetic operators in C

Operator	meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	modulo division

3.2 Arithmetic operators

- Note:,
  - Integer division truncates remainder
  - The % operator cannot be applied to a float or double.
  - The precedence of arithmetic operators
    - Unary + or -
    - * / %
    - + -

3.10 Arithmetic expressions

- An arithmetic expression is a combination of variables, constants, and operators.
- For example,
- a*b-c → a*b-c
- (m+n)(x+y) → (m+n)*(x+y)
- ax^2+bx+c → a*x*x+b*x+c

3.3 Relational Operators

- The relational operators in C are :

Operator	Meaning
<	less that
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Relational Operators

- A relational expression yields a value of 1 or 0.
  - 5 < 6   1
  - -34 + 8 > 23 - 5   0
  - if a=3, b=2, c =1; then a > b > c is ?

- the associativity of relational operators is
  left → right

3.4 Logical operators

- C has the following three logical operators
  - && meaning logical and
  - || meaning logical or
  - ! meaning logical not (unary operator)
- Expressions connected by && or || are evaluated left to right, and evaluation stops as soon as the truth or falsehood of the result is known.

3.5 Assignment operators

- The use of shorthand assignment operators has three advantages:
  - 1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
  - 2. The statement is more concise and easier to read.
  - 3. The statement is more efficient.

3.6 Increment and decrement operators

- C provides two unusual operators for incrementing and decrementing variables.
- The increment operator ++ adds 1 to its operand, while the decrement operator -- subtracts 1.
- The unusual aspect is that ++ and -- may be used either as prefix operators (before the variable, as in ++n), or postfix operators (after the variable: n++).
- In both cases, the effect is to increment n. But the expression ++n increments n *before* its value is used, while n++ increments n *after* its value has been used.

- The increment and decrement operators can be used in complex statements. Example:

  m=n++ -j +10;

- Consider the expression

  m = - n++ ;

- The precedence of ++ and – operators are the same as those of unary + and -.
- The associatively of them is right to left.
- m = - n++; is equivalent to m = - (n++)

3.7 Conditional operator

- a ternary operator pair "? : " is available in C to construct conditional expressions of the form

  expr1 ? expr2 : expr3

- the expression *expr1* is evaluated first. If it is non-zero (true), then the expression *expr2* is evaluated, and that is the value of the conditional expression. Otherwise *expr3* is evaluated, and that is the value. Only one of *expr2* and *expr3* is evaluated.

- z = (a > b) ? a : b;   /* z = max(a, b) */

3.9 Special operators

- **1.The Comma Operator**
- The comma operator can be used to link the related expressions together. A comma-linked list of expressions is evaluated left to right and the value of right-most expression is the value of the combined expression. For example, the statement
- value = (x=10, y=5, x+y);
- first assigns the value 10 to x, then assigns 5 to y, and finally assigns 15 to value. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

3.14 Type conversions in expressions

- **1. Implicit Type Conversion**
- C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance. This automatic conversion is known as implicit type conversion.
- The rule of type conversion: the lower type is automatically converted to the higher type.

- THANKS