by SAHIL

Operators and Expressions

CLASSIFICATION OF OPERATORS

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators

ARITHMETIC OPERATORS

The arithmetic operators supported by are mentioned below.

Operator Symbol	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo Division

1. Integer division truncates any fractional part.
2. The modulo division operator produces the remainder of an integer division.
3. The modulo division operator % cannot be applied on floating point data.

Integer Arithmetic:

1. When both the operands in a single arithmetic expression such as a+b are integers, the expression is called as integer expression and the operation is called integer arithmetic.
2. Integer arithmetic always results into an integer value.
3. During modulo division the sign of result is always the sign of the first operand i.e. dividend.

Example:

Consider a and b are two integer variables with a = 14 and b = 4, with these values of a and b we will get following results for different arithmetic operation.

```
a - b      =  10
a + b      =  18
a  * b     =  56
a  / b     =  3 (decimal part truncated)
a % b      =  2 (remainder of division)
```

RELATIONAL OPERATORS

1. It is often required to compare two quantities and depending on their relation certain decisions are taken.
2. These comparisons are done through relational operators.
3. An expression containing a relational operator is called relational expression.
4. The value of relational expression is either one if the relation is true or zero if the relation is false.
5. C supports six relational operators.

Operator	Meaning
<	is less than
<=	is less than or equal to

Operator	Meaning
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

6. A simple relational expression contains only one relational operator and takes the following form.

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.

Note:

when arithmetic expression are used on either side of a relational operators, the arithmetic expression will be evaluated first and then result are compared.

LOGICAL OPERATORS

C has the following three logical operators.

&&	meaning logical AND
||	meaning logical OR
!	meaning logical NOT

1. The logical operators && and || are used when more than one conditions are to be tested example (a > b && x == 10).
2. An expression combining more than one relational expression is called logical expression or compound relational expression.
3. The logical expressions are also evaluated as one or zero.

ASSIGNMENT OPERATORS

1. ‘ = ’ is known as simple or usual assignment operator.
2. C has a set of shorthand operators.
3. The shorthand operators makes a statement more concise and easier to read.
4. They make a statement more efficient.
5. Some of the commonly used shorthand operators are shown in following table.

Examples:

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

INCREMENT AND DECREMENT OPERATORS

1. C supports two very useful operator ++ and – –.
2. The operator ++ adds 1 to the operand, while – – subtract 1 from the operand.
3. These are called unary operators and takes the general form as:

```
++m;
– –m;
```

Pre-increment and pre-decrement

```
m++;
m– –;
```

Post-increment and post-decrement

4. These operators are generally used in for and while loops.
5. ++m and m++ mean the same when they form statements separately.
6. But they behave differently when they are used in right-hand side of an assignment statement.

Examples:

```
m = 5;
++m;
```

```
m = 5;
m++;
```

Same

```
m = 5;
y = ++m;
```

```
m = 5;
y = m++;
```

Different

BITWISE OPERATORS

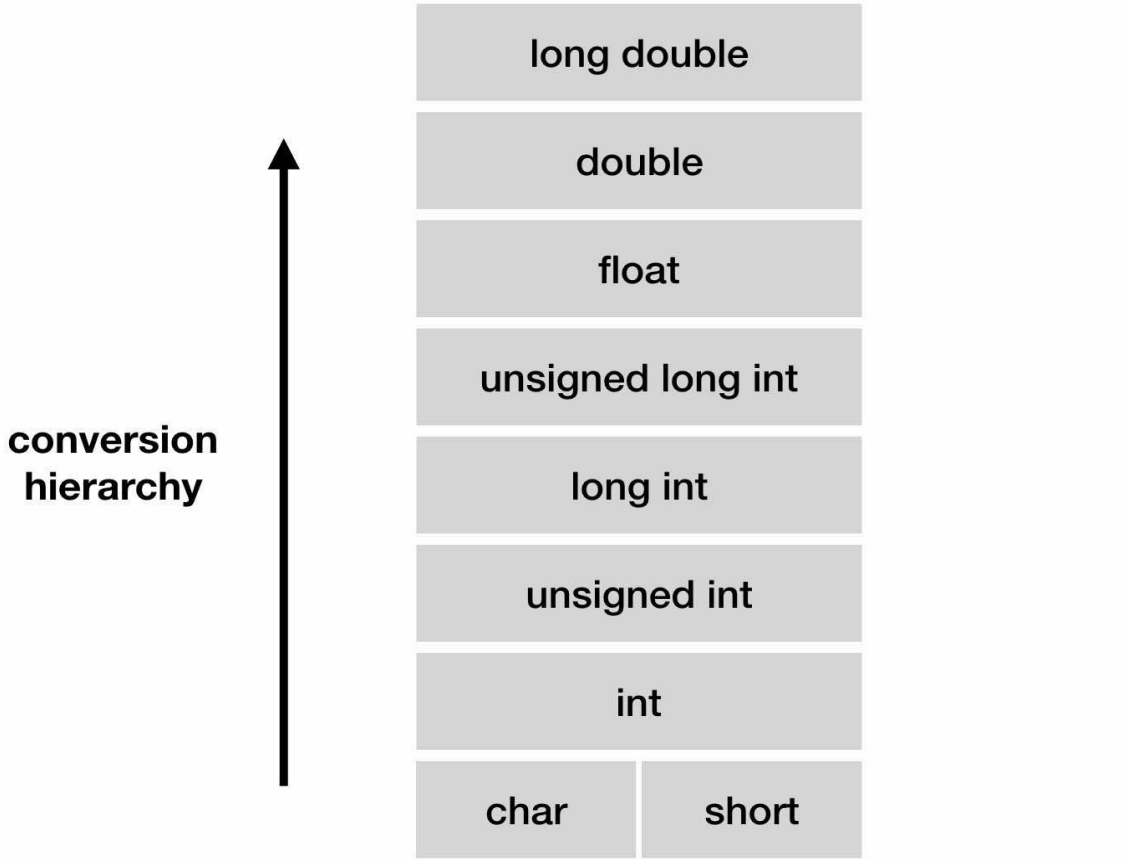
1. C supports manipulation of data at bit level using bitwise operators.
2. Bitwise operators are not applied to float or double.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR(XOR)
<<	Left Shift
>>	Right Shift

TYPE CONVERSION IN EXPRESSION

Implicit Type Conversion:

1. C permits the mixing of constants and variables of different types in an expression.
2. The automatic conversion of a particular value to the proper type is called implicit type conversion.
3. If the operands are of different types, the lower type is automatically converted to the higher type before the operation proceeds, and the result is of the higher type.


conversion
hierarchy
long double
double
float
unsigned long int
long int
unsigned int
int
char
short

Explicit Conversion/Type Casting

1. This type conversion allows a user to convert the type of an operand locally in an expression if required.
2. This type conversion does not affect the value of operand.
3. Also it does not affect the type of operand in the other part of program.
4. It takes the general form as:

```
(type-name) expression
```

5. Example: Consider the calculation of ratio of female to male in a town

```
ratio = female_count / male_count
```

Where female_count and male_count are declared as integer and therefore expression on right hand side will result into integer value, which will represent the wrong result.

6. The above problem can be solved as follows:

```
ratio = (float) female_count / male_count
```

Now the expression on right hand side will be evaluated with floating point mode and the result produced will be of float type.

Use of Casts:

Example	Action
x = (int) 7.5	7.5 is converted to integer by truncation.
a = (int) 21.3/ (int) 5.4	Evaluated as 21/5 and the result would be 4.
b = (double) sum/n	Division is done in floating point mode.
y = (int) (a+b)	The result of a+b is converted to integer.
z = (int) a+b	a is converted to integer and then added to b.
p = cos((double)x)	Converts x to double before using it.

OPERATOR PRECEDENCE AND ASSOCIATIVITY

1. In C each operator has a precedence associated with it.
2. When two or more operators with different precedence level occur in one expression then the operators with higher precedence are evaluated first.
3. The operators with same level of precedence follow the rules of associativity from right to left or left to right in their evaluation.

by SAHIL

Thank You