OPERATORS AND EXPRESSIONS

Various Operators, Precedence and Associativity

Designed by Parul Khurana, LIECA.

Operators

- Individual constants, variables can be joined together by various operators to form expressions.

- The data items that operators act upon are called operands.
  - Some operators require two operands, while other act upon only one operand.

Types of Operators

- Arithmetic
- Unary
- Relational and Logical
- Assignment
- Conditional
- Bitwise
- Special

Types of Operators

- Arithmetic
  - There are 5 arithmetic operators in C. They are:

Operator	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder after integer division

  - The % operator is sometimes referred to as the modulus operator.

Types of Operators

- Arithmetic
  - The operands acted upon by arithmetic operators must represent numeric values.
  - Thus, the operands can be integer quantities, floating-point quantities or characters (*remember that character constants represent integer values, as determined by the computer's character set*).
  - The remainder operator requires that both operands be integers and the second operand be nonzero.
  - Similarly, the division operator (/) requires that the second operator be nonzero.

Types of Operators

- Arithmetic Example
  - Suppose that a and b are integer variables whose values are 10 and 3, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values:

Expression	Value
a + b	13
a – b	7
a * b	30
a / b	3
a % b	1

  - Notice the truncated quotient resulting *from the division operation*, since both operands represent integer quantities.
  - Also, notice the integer remainder resulting from the use of modulus operator in the last expression.

Types of Operators

- Arithmetic Example
  - Suppose that v1 and v2 are floating-point variables whose values are 12.5 and 2.0, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values:

Expression	Value
v1 + v2	14.5
v1 – v2	10.5
v1 * v2	25.0
v1 / v2	6.25

  - **The operation v1 % v2 is illegal and reported as error by the compiler.**

Types of Operators

- Arithmetic Example
  - Suppose that c1 and c2 are character-type variables that represent the characters 'P' and 'T', respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values (based upon the ASCII character set).

Expression	Value
c1	80
c1 + c2	164
c1 + c2 + 5	169
c1 + c2 + '5'	217

  - Note that P is encoded as (decimal) 80, T is encoded as 84, and 5 is encoded as 53 in the ASCII character set.

Types of Operators

- Arithmetic – Conversion Rules
  - Operands that differ in type may undergo type conversion before the expression takes on its final value.
  - In general, the final result will be expressed in the highest precision possible, *consistent with the data types of operands*.

Types of Operators

- Arithmetic Operations – Conversion Rules
  - If one of the operands is long double, the other will be converted to long double and the result will be long double.
  - Otherwise, If one of the operands is double, the other will be converted to double and the result will be double.
  - Otherwise, If one of the operands is float, the other will be converted to float and the result will be float.

Types of Operators

- Arithmetic Operations – Conversion Rules

  - If one of the operands is unsigned long int, the other will be converted to unsigned long int and the result will be unsigned long int.

  - Otherwise, If one of the operands is long int and the other is unsigned int, then:
    - If unsigned int, can be converted to long int, the unsigned int operand will be converted as such and the result will be long int.
    - Otherwise, both operands will be converted to unsigned long int and the result will be unsigned long int.

Types of Operators

- Arithmetic Operations – Conversion Rules
  - Otherwise, If one of the operands is long int, the other will be converted to long int and the result will be long int.
  - Otherwise, If one of the operands is unsigned int, the other will be converted to unsigned int and the result will be unsigned int.
  - If none of the above conditions applies, then both operands will be converted to int(if necessary), and the result will be int.

Types of Operators

- Unary
  - C includes a class of operators that act upon a single operand to produce a new value. Such operators are known as unary operators.
  - Unary operators usually precede their single operands, though some unary operators are written after their operands.

Types of Operators

- Unary
  - Perhaps the most common unary operation is unary minus, where a numerical constant, variable or expression is preceded by a minus sign.
    - Some programming languages allow a minus sign to be included as a part of a numeric constant. In C, however, all numeric constants are positive.
    - *Thus, a negative number is actually an expression, consisting of the unary minus operator, followed by a positive numeric constant*.

Types of Operators

- Unary
  - Note that the unary minus operation is distinctly different from the arithmetic operator which denotes subtraction (-). *The subtraction operator requires two separate operands.*
  - *Here are several examples which illustrate the use of the unary minus operation,*

-743	-0X7FFF	-0.2	-5E-8
-root1	-(x + y)	-3 * (x + y)	

Types of Operators

- Unary
  - There are two other commonly used unary operators:
    - The increment operator, **++**, and the decrement operator, **--**.
    - The increment operator causes its operand to be increased by 1, whereas the decrement operator causes its operand to be decreased by 1.
      - *The operand used with each of these operators must be a single variable.*

Types of Operators

- Unary – Example

Suppose that **i** is an integer variable that has been assigned a value of **5**. The expression **++i**, which is equivalent to writing **i = i + 1**, causes the value of **i** to be increased to **6**.

Similarly, the expression **--i**, which is equivalent to **i = i – 1**, causes the (original) value of **i** to be decreased to **4**.

Types of Operators

- Unary
  - The increment and decrement operators can each be utilized in two different ways, depending on whether the operator is written before or after the operand.

Types of Operators

- Unary
  - If the operator precedes the operand (e.g., ++i), then it is called as pre-increment operator and the operand will be altered in value before it is utilized for its intended purpose within the program.
  - If, however, the operator follows the operand (e.g., i++), then it is known as post-increment operator and the value of the operand will be altered after it is utilized.

Types of Operators

- Unary - Example
  - The increment/decrement operator works as simple increment/decrement by one, if it is used not as a part of an expression.
  - In that case, there is no difference between operation of pre or post increment/decrement operator.

Types of Operators

- Unary - Example
  - For example consider the following statements:

```
a=10;
a++;
```

  - This post-increment simply increments value of a by 1, i.e., the value of a becomes 11 after it.
    - Even if this post increment operator is replaced with pre increment operator, there **will be no change** in its **output** or working.

Types of Operators

- Unary - Example
  - However, when these operators are used as a part of an expression then the difference can be noticed, e.g., consider the following set of statements:

```
a=10; b = 10;
x = ++a;
y = b++;
printf( " x = %d  a = %d \n",  x,  a  );
printf( " y = %d  b = %d \n",  y,  b  );
```

Here **x = ++a** statement is equivalent to following two statements in sequence:

a = a + 1;

x = a;

Thus, the value of both x and a will be **11**.

Types of Operators

- Unary - Example
  - However, when these operators are used as a part of an expression then the difference can be noticed, e.g., consider the following set of statements:

```
a=10; b = 10;
x = ++a;
y = b++;
printf( " x = %d  a = %d \n",  x,  a  );
printf( " y = %d  b = %d \n",  y,  b  );
```

On the other hand, **y = b++** is equivalent to:

y = b;

b = b + 1;

So, the value of y will be **10** and b will be **11**.

Types of Operators

- Relational and Logical
  - There are 4 relational operators in C. They are:

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

  - These operators all fall within the same precedence group, which is lower than the arithmetic and unary operators.
  - The associativity of these operators is left to right.

Types of Operators

- Relational and Logical
  - Closely associated with the relational operators are the following two equality operators.

Operator	Meaning
==	equal to
!=	not equal to

  - The equality operators fall into a separate precedence group, beneath the relational operators.
  - These operators also have a left-to-right associativity.

Types of Operators

- Relational and Logical
  - These 6 operators are used to form logical expressions, which represent conditions that are either true or false.
  - The resulting expression will be of type integer, since true is represented by the integer value 1 and false is represented by the value 0.

Types of Operators

- Relational and Logical - Example
  - Suppose that i, j and k are integer variables whose values are 1, 2 and 3, respectively. Several logical expressions involving these variables are shown below:-

Expression	Interpretation	Value
i < j	true	1
(i + j) >= k	true	1
(j + k) > (i + 5)	false	0
k ! = 3	false	0
j == 2	true	1

Types of Operators

- Relational and Logical
  - In addition to the relational and equality operators. C contains two logical operators (also called logical connectives). They are:

Operator	Meaning
&&	and
||	or

  - These operators are referred to as **logical and** and **logical or**, respectively.

Types of Operators

- Relational and Logical
  - The logical operators act upon operands that are themselves logical expressions.
  - The net effect is to combine the individual logical expressions into more complex conditions that are either true or false.
  - The result of a **logical and** operation will be true **only if both operands are true**, whereas the result of **logical or** operation will be true **if either operand is true or if both operands are true**.
  - In other words, the result of logical or operation will be **false** only if both operands are false.

Types of Operators

- Relational and Logical - Example
  - Suppose that i is an integer variable whose value is 7, f is floating-point variable whose value is 5.5, and c is a character variable that represents the character 'w'. Several complex logical expressions that make use of these variables are shown below:

Expression	Interpretation	Value
(i >= 6) && (c == 'w')	true	1
(i >= 6) || (c == 119)	true	1
(f < 11) && (i > 100)	false	0
(c != 'p') || ((i + f) <= 10)	true	1

Types of Operators

- Relational and Logical - Example
  - The first expression is true because both operands are true. In the second expression, both operands are again true; hence the overall expression is true. The third expression is false because the second operand is false. And finally, the fourth expression is true because the first operand is true.

Expression	Interpretation	Value
(i >= 6) && (c == 'w')	true	1
(i >= 6) || (c == 119)	true	1
(f < 11) && (i > 100)	false	0
(c != 'p') || ((i + f) <= 10)	true	1

Types of Operators

- Relational and Logical
  - C also includes the unary operator ! that negates the value of a logical expression; i.e., it causes an expression that is originally true to become false, and vice versa.
  - This operator is referred to as the **logical negation (or logical not) operator.**

Types of Operators

- Relational and Logical - Example
  - Suppose that i is an integer variable whose value is 7, f is floating-point variable whose value is 5.5. Several logical expressions that make use of these variables and the logical negation operator are shown below:

Expression	Interpretation	Value
f > 5	true	1
! (f > 5)	false	0
i <= 3	false	0
! (i <= 3)	true	1
i > (f + 1)	true	1
! (i > (f + 1))	false	0

Types of Operators

- Assignment
  - There are several different assignment operators in C.
  - All of them are used to form assignment expressions, which assign the value of an expression to an identifier.
  - The most commonly used assignment operator is =.

Types of Operators

- Assignment
  - Assignment expressions that make use of this operator are written in the form:

Identifier = expression

Where identifier generally represents a variable, and expression represents a constant, a variable or a more complex expression.

Types of Operators

- Assignment - Example
  - Here are some *typical expressions* that make use of the = operator.

```
a = 3
x = y
delta = 0.001
sum = a + b
area = length * width
```

The **first assignment** expression causes the integer value 3 to be assigned to the variable a, and the **second assignment** causes the value of y to be assigned to x. In the **third assignment**, the floating-point value 0.001 is assigned to delta. The **last two assignments** each result in the value of an arithmetic expression to a variable.

Types of Operators

- Assignment
  - Remember that the assignment operator = and the equality operator == are different.
  - **The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two expressions have the same value.**
  - **These operators cannot be used in the place of one another.**

Types of Operators

- Conditional
  - Simple conditional operations can be carried out with the conditional operator (? :).
  - An expression that makes use of the conditional operator is called a conditional expression.
  - Such an expression can be written in place of the more traditional *if-else statement*.

Types of Operators

- Conditional
  - *A conditional expression is written in the form:*

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

  - When evaluating a conditional expression, expression 1 is evaluated first.
    - If expression 1 is true (i.e., if its value is nonzero), then expression 2 is evaluated and **this becomes the value of the conditional expression**.
    - However, if expression 1 is false (i.e., if its value is zero), then expression 3 is evaluated and **this becomes the value of the conditional expression**.

Types of Operators

- Conditional - Example
  - In the conditional expression shown below, assume that i is an integer variable.

```
( i < 0 )   ?     0     :     100
```

  - The expression (i < 0) is evaluated first. If it is true (i.e., the value of i is less than 0), *the entire conditional expression takes on the value 0*.
  - Otherwise (if the value of i is not less than 0), the entire conditional expression takes on the value 100.

Precedence and Associativity

- Precedence
  - The operators within C are grouped hierarchically according to their precedence (*i.e., order of evaluation*).
  - Operations with higher precedence are carried out before operations having a lower precedence.

Types of Operators

- Bitwise
  - These operators perform bit-level operations on integer operands.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
~	Bitwise COMPLIMENT
^	Bitwise XOR
<<	Bitwise SHIFT - LEFT
>>	Bitwise SHIFT - RIGHT

Types of Operators

- Special

Operator	Purpose
sizeof	Returns number of bytes allocated to given value or data-type.
&	Returns the address of given variable.
(type)	Explicit Type Casting
.	Member Access (Dot Operator)
->	Member Access through pointer (Arrow Operator)

Precedence and Associativity

- Associativity
  - Another important consideration is the order in which consecutive operations within the same precedence group are carried out.

    - ***This is known as associativity*.**

Precedence and Associativity

- *The precedence groups are listed from highest to lowest.*

Precedence Group	Operators	Associativity
function, array, structure member, pointer to structure member	() [] . ->	L -> R
unary operators	- ++ -- ! - * & sizeof (type)	R -> L
arithmetic multiply, divide and remainder	* / %	L -> R
arithmetic add and subtract	+ -	L -> R
bitwise shift operators	<< >>	L -> R
relational operators	< <= > >=	L -> R
equality operators	== !=	L -> R

Precedence and Associativity

- *The precedence groups are listed from highest to lowest.*

Precedence Group	Operators	Associativity
bitwise and	&	L -> R
bitwise exclusive or	^	L -> R
bitwise or	|	L -> R
logical and	&&	L -> R
logical or	||	L -> R
conditional operator	? :	R -> L
assignment operators	= += -= /= %= &= ^= |= <<= >>==	R -> L
comma operator	,	L -> R

Practice Questions

- Find the result of the following expressions:
  - 5 + 7 * 30 / 8 + 6.4
  - 7 < 5 && 3 < 7
  - 20 > 45 & 4
  - 35 << 4 * 4
  - 047 ^ 0x25
  - 22 % 3 * 12 / sizeof (3)
  - 200 / 30 / 3 / 2
  - sizeof (“CSE”) + (char) 101
  - 6 && 7 || -1 && 10