Lecture 3. Operators

**Expressions**

* An expression is a combination of variables, constants, and operators
* Expressions are written according to the syntax of C language
* In a statement:

variable = expression;

the expression is evaluated first and then the previous value of the variable on the left is replaced.

* The =, +=, -=, \*=, /=, and %= operators are always applied last in an expression.

Examples:

num1 = num2

a + b

x = y + z

price <= 100

++counter

# Operators

C operators can be classified into following types:

* Arithmetic operators
* Relational operators
* Logical operators
* Bitwise operators
* Assignment operators
* Conditional operators
* Special operators

## Arithmetic Operators

* Arithmetic operators
  + **Unary** – require only one operand: positive (+a), negative (-a), increment (a++, ++a), decrement (a--, --a)
  + **Binary** – require two operands: +, -, \*, /, %

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| + | Adds two operands. | A + B |
| − | Subtracts second operand from the first. | A − B |
| ∗ | Multiplies both operands. | A ∗ B |
| ∕ | Divides numerator by de-numerator. | B ∕ A |
| Division rules: int/int = int float/float = float float/int = float int/float = float |
| % | Modulus Operator  (finds the remainder after division) | B % A 9%2 = 1 |
| ++ | Increment operator increases the integer value by one. | A++ is equivalent to A=A+1  ++A is equivalent to A=A+1 |
| -- | Decrement operator decreases the integer value by one. | A— is equivalent to A=A-1  --A is equivalent to A=A-1 |

## Pre-/Post- Increment & Decrement Operators

* Change value of a variable before (prefix mode) or after (postfix mode) its value is used in an expression.
* The following table illustrates the difference between pre-/post- increment and decrement: Example: int result, num = 5;

|  |  |  |  |
| --- | --- | --- | --- |
| Statement | Order of operations | **result** value | **num** value |
| result = num++; | result = num;  num = num +1; | 5 | 6 |
| result = ++num; | num = num +1;  result = num; | 6 | 6 |
| result = num--; | result = num;  num = num - 1; | 5 | 4 |
| result = --nun; | num = num -1;  result = num; | 4 | 4 |

## Assignment Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | Simple assignment operator. Assigns values from right side operands to left side operand. | C = A + B |
| += | Add AND assignment operator.  Adds the right operand to the left operand and assigns the result to the left operand. | C += A  is equivalent to C = C + A |
| −= | Subtract AND assignment operator.  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.  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.  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.  Calculates modulus using two operands and assigns the result to the left operand. | C %= A  is equivalent to C = C % A |

**Relational Operators**

* Used to check the relationship between two operands (two values).
* Have two results TRUE or FALSE:
* If the relationship is TRUE, it returns 1
* If the relationship is FALSE, it returns 0
* Used in decision making and loops
* Arithmetic operators have higher priority than relational operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| == | Checks if the values of two operands are equal or not.  Returns 1 (TRUE) if two operands are equal. Otherwise, returns 0 (FALSE). | (8 == 3)  returns 0 |
| != | Checks if the values of two operands are equal or not. Returns 1 (TRUE) if two operands are NOT equal. Otherwise, returns 0 (FALSE). | (8 != 3)  returns 1 |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes TRUE and 1 is returned. Otherwise, 0 is returned (FALSE). | (8 > 3)  returns 1 |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes TRUE and 1 is returned. Otherwise, 0 is returned (FALSE). | (8 < 3)  returns 0 |
| >= | 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 and 1 is returned. Otherwise, 0 is returned (FALSE). | (8 >= 3)  returns 1 |
| <= | 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 and 1 is returned. Otherwise, 0 is returned (FALSE). | (8 <= 3)  returns 0 |

Examples:

number1 >= number2 number1 == number2 number1 != 5

## Logical Operators

### Used when more than one condition needs to be tested.

* Expressions containing logical operators return 0 or 1 depending on expression result – 0 if the expression result is FALSE, 1 if the expression result is TRUE.
* Used in decision making.

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example A=1, B=0 |
| && | Logical AND operator. If both the operands are non-zero (TRUE), then the condition becomes TRUE. | (A && B)  is FALSE (0) |
| || | Logical OR Operator. If any of the two operands is non-zero (TRUE), then the condition becomes TRUE. | (A || B)  is TRUE (1) |
| ! | Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is TRUE, then Logical NOT operator will make it FALSE. | !(A && B) is TRUE (1) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | A && B | A || B | ! A |
| False | False | False | False | True |
| False | True | False | True | True |
| True | False | False | True | False |
| True | True | True | True | False |

Examples:

number1 >= 20 && number2 == 20

!(number1 < 5 && number2 > number1)

## Operator Precedence

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Priority** |
| ( )  [ ] | Parentheses (function call) Brackets (array subscript) | Highest |
| . | Member selection via object name |  |
| -> | Member selection via pointer |  |
| ++ – | Postfix increment/decrement (see Note 2) |  |
| ++ – | Prefix increment/decrement |  |
| + – | Unary plus/minus |  |
| ! ~ | Logical negation/bitwise complement |  |
| (*type*) | Cast (convert value to temporary value of *type*) |  |
| \* | Dereference |  |
| & | Address (of operand) |  |
| sizeof | Determine size in bytes on this implementation |  |
| \* / % | Multiplication/division/modulus |  |
| + – | Addition/subtraction |  |
| << >> | Bitwise shift left, Bitwise shift right |  |
| < <= | Relational less than/less than or equal to |  |
| > >= | Relational greater than/greater than or equal to |  |
| == != | Relational is equal to/is not equal to |  |
| & | Bitwise AND |  |
| ^ | Bitwise exclusive OR |  |
| | | Bitwise inclusive OR |  |
| && | Logical AND |  |
| | | | Logical OR |  |
| ? : | Ternary conditional |  |
| = | Assignment |  |
| += -= | Addition/subtraction assignment |  |
| \*= /= | Multiplication/division assignment |  |
| %= &= | Modulus/bitwise AND assignment |  |
| ^= |= | Bitwise exclusive/inclusive OR assignment |  |
| <<= >>= | Bitwise shift left/right assignment |  |

|  |  |  |
| --- | --- | --- |
| , | Comma (separate expressions) | Lowest |

**References**

* Tan, H.H., and T.B. D’Orazio. *C Programming for Engineering & Computer Science*. USA: WCB McGRaw-Hill. 1999. Print.
* Tutorialspoint.com. "C Operators." *Www.tutorialspoint.com*. N.p., n.d. Web. 01 Feb. 2017.

<<https://www.tutorialspoint.com/cprogramming/c_operators.htm>>.

* "C Programming Operators." *C Operators: Arithmetic, Logical, Conditional and more*. N.p., n.d. Web. 01 Feb. 2017. <<https://www.programiz.com/c-programming/c-operators>>.
* Rajinikanth. "C Programming Language." *C Operators | c operators | operators in c | C by Rajinikanth | C Programming Language*. N.p., n.d. Web. 09 Feb. 2017.

<<http://www.btechsmartclass.com/CP/c-operators.htm>>.