# Operators

ESC108A Elements of Computer Science and Engineering

B. Tech. 2017

**Course Leaders:**

**Roopa G.**

**Ami Rai E.**

**Chaitra S.**

## Objectives

- At the end of this lecture, student will be able to
  - Explain operators in C
  - Use different types of operators

# Contents

- Operators
- Types of operators
- Operator precedence and associativity

# Operators and Operands

- Expressions consist of variables, operators and operands

  a = b + 5

- Operator: Symbol representing the operation

  = and +

- Operand: The data items (variables and constant) on which the operation is performed

  a, b and 5

  - In case of + operator, operands are b and 5
  - In case of = operator, operands are a and the value of expression b+5

## Operators – Types

- Operators are classified based on number of operands as
  - Unary operators
  - Binary operators
  - Ternary operators

# Assignment Operator

- Assignment Operator =

  a = 1;

  1 is assigned to the variable a

- Do not confuse with equality Operator ==

## Arithmetic Operators

+ (addition)

    a + 5

- (subtraction)

    10 - b

* (multiplication)

    a * c

/ (division) – Integer division

    a / 5            Example: 10/5 = 2   and  35/2 = 17

% (modulo) - Remainder

    a % 2            Example: 35%2 = 1

## Compound Assignment Operators

- Any statement of the form

  variable = variable operator expression;

- Can be written as

  variable operator = expression;

+= ,-=, *=, /=, %=

```
a = a + 1;
```

can be written as

```
a += 1;
```

# Increment and Decrement Operators

- Unary operators for increment and decrement

  ++

  a++; /*post increment - Increment value of a after this expression is evaluated*/

  ++a; /* pre increment - Increment value of a first and then evaluate the expression */

  --

  a--; /* post decrement - Decrement value of a after this expression is evaluated*/

  --a; /* pre decrement - Decrement value of a first and then evaluate the expression */

# Increment and Decrement Operators

- Let a=5 and b=10

```
a++;          //a becomes 6
b--;          //b becomes 9
```

- Example

```
int a,b,x=10,y=10;
a = x++; b = ++y;
printf("Value of a : %d",a);
printf("Value of b : %d",b);
```

- Applicable only to variables
  - n++        //legal
  - (i+j)++  //illegal

## Operators for Comparison

- Relational Operators
  - < Less than
  - > Greater than
  - != Not equal to
  - == Is it equal to? (Do not confuse with assignment opoerator =)
  - <= Less than or equal to
  - >= Greater than or equal to
- Return
  - 0 if false
  - any other number if true (generally 1 is returned on true)

## Logical Operators

- Used to take decisions

  && logical AND

  ```
  if ((a<b) && (a<c)) {}
  ```

  || logical OR

  ```
  if ((a<b) || (a<c)) {}
  ```

  ! logical NOT (unary operator)

  ```
  if(! ((a<b) && (a<c))) {}
  ```

- Return values
  - 0 if false
  - 1 if true (generally)

## Question

- How can we get the bits in memory?

# Bitwise Operators

- All data items are stored in computer's memory as a sequence of bits(0s and 1s)

  & Bitwise AND The bits in the result are set to 1 if the corresponding bits in the two operands are both 1 E.g., a&b;

  | Bitwise OR The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1 E.g., a|b;

  – Suppose a and b are 8-bit Integers

    a has value 11010011

    b has value 10101100

    a&b ?

    a|b ?

# Bitwise Operators contd.

! Complement All 0 bits are set to 1 and all 1 bits are set to 0

E.g., c = !a;

^ Bitwise XOR The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1

E.g., c = a^b;

<< Left shift Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits

E.g., a<<2;

>> Right Shift Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent

E.g., a>>2;

# Conditional Operators

- Ternary operator  ?:

- It has the general form

  exp1 ? exp2 : exp3

  exp1 is evaluated. If it is true, then exp2 is evaluated and becomes the value of the expression. If exp1 is false, then exp3 is evaluated, and that is the value

- It takes 3 operands
- Conditional expression
  - Conditional operators with the operands

## Conditional Operators - Example

```
main(){
   int num;
   printf("Enter the Number : ");
   scanf("%d",&num);
   (num%2==0)?printf("Even"):printf("Odd");
}
```

# sizeof() Operator

- sizeof() operators returns the size (number of bytes) of the operand occupies
- Must precede its operand
- Operand may be a constant, a variable or a data type
- Syntax

```
sizeof(operand);
```

- Example

```
x=sizeof(int);
y=sizeof(x);
```

# Operators on Floating Point Data Type

- Operators that can be used on ***float*** and ***double*** data types
  - All Arithmetic operators except %(modulo) operator
  - All Comparison operators
  - Compound Assignment operators
  - sizeof() operator

# Operator Precedence and Associativity

- Consider

```
a = 1+2*3/4;
```

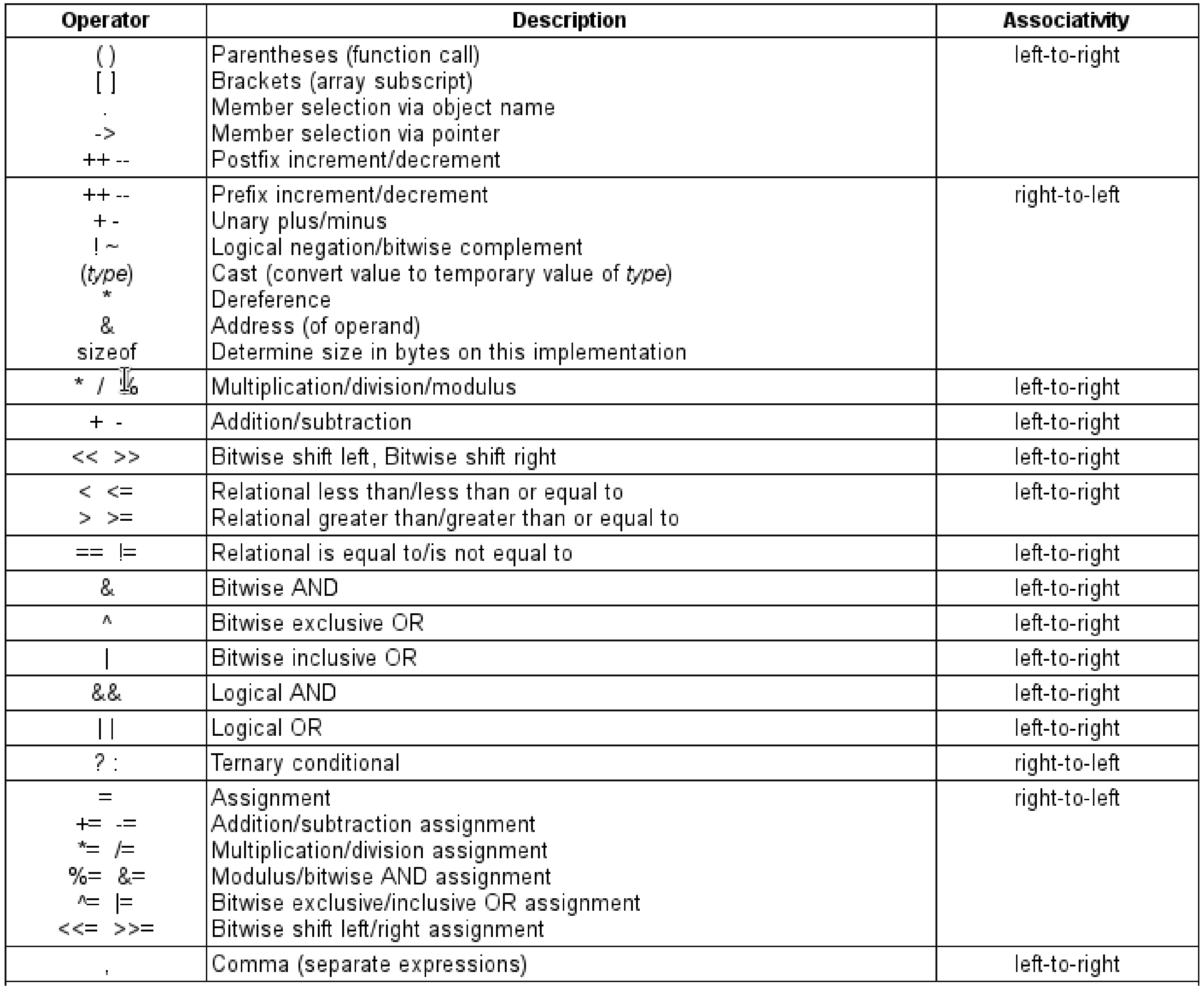
- What is the value of a?

## Operator Precedence and Associativity

- To avoid confusion while reading, always use brackets
- The computer always gives one answer only as it understands expressions based on precedence and associativity

- Precedence
  - Which operator to evaluate first
- Associativity
  - Which operand to evaluate first

# Operator Precedence and Associativity



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

C Operator Precedence Table (DiFranco 2011)

## Example

- Consider

```
a = 1+2*(3/4);

Treated equal to a = 1+(2*(3/4) );
3/4
2*0
1+0
1

a = 1
```

- Refer to the operator precedence chart when writing expressions containing many operators
- If you are uncertain about the order of evaluation in a complex expression, use parenthesis to group expressions

# Summary

- Expressions consist of
  - operators, the symbols that represent an operation
  - operands, the data items on which the operation is applied
- There are many types of operators
  - Arithmetic
  - Comparison
  - Logical
  - Bitwise
  - Conditional, etc.,
- Expressions are evaluated based on precedence and associativity of operators

## References

DiFranco, D. (2011) *C Operator Precedence Table,* available at *http://www.difranco.net/compsci/C_Operator_Precedence_Table.htm (accessed 28 July 2014).*

## Further Reading

Kernighan, B. W. and Richie, D. (1992) *The C Programming Language.* 2nd ed., New Delhi:PHI.