Lecture-3.1

Contents

- Operator & Expression
  - Arithmetic operator
  - Relational operator
  - Logical operator
  - Conditional operator
  - Bitwise operator
  - Increment and decrement operator
  - Assignment operator
  - *sizeof* operator
- Operator precedence & associativity
- Type conversion

Operators and Expression

- Operator
  - Symbol that tell the computer to perform certain mathematical or logical manipulations
  - Used in programs to manipulate data
  - Form a part of the mathematical or logical expression
  - Example: +, *, %, =, etc.
- Expression
  - Sequence of operands and operators that reduces to a single value
  - Example, 5+3 is an expression whose value is 8.

Arithmetic operator

Let, A = 10 and B = 5

Operator	Meaning	Expression	Value
+	Addition	A + B	15
-	Subtraction	A – B	5
*	Multiplication	A * B	50
/	Division	A / B	2
%	Modulo (Remainder after division)	A % B	0

 All of these arithmetic operators are binary operator as they take 2 operands

Relational operator

Let, A = 10 and B = 5

Operator	Meaning	Expression	Value
==	Equal to	A == B	0
!=	Not Equal to	A != B	1
>	Greater than	A > B	1
<	Less than	A < B	0
>=	Greater than or Equal to	A >= B	1
<=	Less than or Equal to	A <= B	0

- All of these relational operators are binary operator as they take 2 operands
- Interpretation: **1** means **True** and **0** means **False**

Logical operators

Let, A = 10 and B = 5

Operator	Meaning	Expression	Value
&&	Logical AND	(A>B) && (A<100)	1
||	Logical OR	(A>B) || (A>100)	1
!	Logical NOT	! (A>B)	0

- **&&** and **||** are binary operator as they take 2 operands
- **!** is an Unary operator as it takes one operand
- Interpretation: **1** means **True** and **0** means **False**

Conditional operator

- **? :**
- Operand 1 **?** Operand 2 **:** Operand 3
- Syntax
  - *test expression* **?** *Expression 1* **:** *Expression 2* **;**
- *Example*
  - *(B%2) == 0 ? printf(“even”) : printf(“odd”);*
  - Output ???
  - *A = (B%2) == 0 ? printf(“even”) : printf(“odd”);*
  - *printf(“%d”,A);*
  - Output ???
- It is a ternary operator

Bitwise operator

Let, A = 00010001 and B = 11000011 (in binary)

Operator	Meaning	Expression	Value(in binary)
~	Bitwise negation one's complement	~ A	11101110
&	Bitwise AND	A & B	00000001
|	Bitwise OR	A | B	11010011
^	Bitwise exclusive OR (XOR)	A ^ B	11010010
>>	Right shift by RHS operand (Divide by power of 2)	A >> 2	00000100
<<	Left shift by RHS operand (Multiply by power of 2)	A << 2	01000000

- **~** is an Unary operator as it takes one operand
- All other operators here, are Binary operator as they take 2 operands
- Interpretation: **1** means **True** and **0** means **False**

Increment and decrement operator

- **a + +** □ postfix increment operator [a = a + 1]
- **+ + a** □ prefix increment operator [a = a + 1]
- **a - -** □ postfix decrement operator [a = a – 1]
- **- - a** □ prefix decrement operator [a = a – 1]

- All of these are Unary operator as they take only one operand

Increment and decrement operator

Example

```
main(){
      int a=5;
      a++;
      printf("%d\n",a);
      printf("%d\n",a++);
      printf("%d\n",++a);
      printf("%d\n",a--);
      printf("%d\n",--a);
}
```

Output ???

Increment and decrement operator

Example

```
main(){
      int a=5;
      (A+B)++;
}
```



Is it possible or valid statement ???

Assignment operator

Operator	Example	Same as
=	A = B	A = B
+ =	A += B	A = A + B
- =	A -= B	A = A – B
* =	A *= B	A = A * B
/ =	B /= A	B = B / A
% =	A %= B	A = A % B

All of these are binary operator as they take 2 operands

sizeof operator

- *Sizeof* operator returns the size of its operand in bytes
- The operand may be an expression, or it may be a cast
- Used to determine length of arrays and structures when their size are unknown to the programmer
- Used to allocate memory space dynamically to variables during the execution period of a program (at run time)
- Example

```
main(){
        int a=5;
        printf("Size of variable a is %d\n",sizeof(a));
        printf("Size of integer is %d\n",sizeof(int));
        printf("Size of floating-point is %d\n",sizeof(float));
}
```

Output ????

Operator precedence and assosiativity

- Operator precedence
  - The measurement of how an expression is evaluated
  - 2 distinct levels of precedence
    - Higher
    - Lower
  - An operator may belong to any one of these levels
  - Operators with highest precedence are evaluated first

High precedence: *, /, %

Low precedence: +, -

Operator precedence and assosiativity

- Assosiativity
  - Operators with same precedence are evaluated either from ‘left to right’ or from ‘right to left’, depending on the level, this is known as associativity property of an operator.

- Rules of precedence and assosiativity

Type Conversion

- Implicit type conversion
  - Hierarchical conversion from lower size type to higher size type
- Explicit type conversion
  - Type casting
  - Syntax: *(type) operand*
  - *Example:*

```
int a;
Printf("%f", (float) a);
```

