Operator, Operand, and Arithmetic

1

Sub Topics

Operator, Operand, and Arithmetic:

- Operator and Operand Introduction
- Assignment Operators
- Arithmetic Operators
- Relational Operators
- Conditional Expressions
- Logical Operators
- Bitwise Operators
- Pointer Operators
- Precedence and Associative
- Exercise

2

Operator and Operand Introduction

- **Operator** is a symbol to process values in result for a new value
- **Operand** is part which specifies what data is to be manipulated or operated on
- Example :
  C = A + B
  (= and + sign are **operators**, A, B and C are **operands**)
- Based on its operand number, operator can be divided into three:
  - **Unary operator** (needs one operand)
  - **Binary operator** (needs two operands)
  - **Ternary operator** (needs three operands)

3

Operator and Operand Introduction

- Based on its operation type, **operator** can be grouped as:
  - Assignment Operator
  - Logical Operator
  - Arithmetic Operator
  - Relational Operator
  - Bitwise Operator
  - Pointer Operator

4

Assignment Operators

- A binary operator
- Used in assigning value to an operand
  Syntax:
  Operand1 = Operand2;
- Left hand side operand (**Operand1**) should have (L-Value) such as variable
- Right hand side operand (**Operand2**) can be constant, another variable, expression or function

5

Assignment Operators

- Example:

```
x = 2;              // constant
x = y;              // other variable
x = 2 * y;          // expression
x = sin (y);        // function
```

- Type of the result will follow the left hand side operand

```
int x = 7/2;      /*x value is 3 not 3.5*/
float y = 3;      /*y value is 3.000 */
```


6

Arithmetic Operators

Symbol	Functionality	Example
+	Addition	x = y + 6;
-	Subtraction	y = x – 5;
*	Multiply	y = y * 3;
/	Division	z = x/y;
%	Modulo	A = 10 % 3;
++	Increment	x++;
--	Decrement	z--;
()	Scope / Priority	x=(2+3)*5

7

Arithmetic Operators

- **Modulo**
  - Symbol : %
  - Binary operator
  - To find remainder of a division
  - N % 2, can be used to find an odd or even number
    - N % 2 = 0 → N is even
    - N % 2 = 1 → N is odd
- **Increment and Decrement**
  - Symbol : ++(increment), --(decrement)
  - Unary operator
  - Increase (++) and decrease (--) the value of a variable by 1.
  - Its position can be in the front (pre) or after (post) a variable.

8

Arithmetic Operators

- Example :

```
N++;   // post increment
++N;   // pre increment
N--;   // post decrement
--N;   // pre decrement
```

- If a stand alone statement, **`N++;`** or **`++N;`** equal to **`N=N+1;`**
- If a stand alone statement, **`N--;`** or **`--N;`** equal to **`N=N-1;`**

9

Arithmetic Operators

- Example:

```
#include <stdio.h>
int main ()
{
       int x = 44;  int y = 44;
       ++x;
       printf("x = %d\n", x);     /* result 45 */
       y++;
       printf("y = %d\n", y);     /* result 45 */
}
```


10

Arithmetic Operators

- If **++n** and **n++** as a bounded statement (sub expression), then both of them have a different meaning
- **++n** → n add by 1, then continue to process its expression
- **n++** → process the expression initially, than add **n** by 1

```
int main () {
       int x=44; int y = 44; int z;
       z =  ++x;      /* z, x is 45 */
       z =  y++;      /* z is 44 and y is 45 */
       return(0);
}
```


11

Arithmetic Operators

Every expression using the following form:

<Variable> = <Variable> <Operator><Exp>;

has similar meaning with:

<Variable> <Operator> = <Exp>;

or also known as:

Combined Operator

Expression	Combined Operator
a = a + b;	a += b;
a = a – b;	a -= b;
a = a * b;	a *= b;
a = a / b;	a /= b;
a = a % b;	a %= b;
a = a ^ b ;	a ^= b;

12

Arithmetic Operators

Example :

x *= y +1; has similar meaning with:
A. x = x * (y + 1);
B. x = x * y + 1;
C. x = x + 1 * y;
D. x = (x + 1) * y;

Answer: A

13

Relational Operators

Use to compare to values with TRUE or FALSE result

Symbol	Functionality
==	Equality
!=	Not equal
<	Less than
>	Greater than
<=	Less or equal than
>=	Greater or equal than
?:	Conditional assignment

FALSE in C language equals to the value of **zero**
TRUE on the other hand not equal to zero
TRUE set by a C program at run time equal to the value of **one**

14

Relational Operators

Example:

```
#include<stdio.h>
int main()
{
      int x=5,y=6;
      if ( x == y) printf("%d equal %d\n",x,y);
      if ( x != y) printf("%d not equal %d\n",x,y);
      if ( x < y) printf("%d less than %d\n",x,y);
      if ( x > y) printf("%d greater than %d\n",x,y);
      if ( x <= y) printf("%d less or equal than %d\n",x,y);
      if ( x >= y) printf("%d greater or equal than %d\n",x,y);
      return(0);
}
```

```
    int x;
    x = (20 > 10);     // x value 1
    x = (20 == 10);    // x value 0
```


15

Conditional Expressions

- Given the following statement:
  if(a > b) z = a;
  else z = b;
- The above statement can be reformed to a **conditional expression**
- Conditional expression using **ternary operator** : '?' and ':'
- Syntax :
  exp1 ? exp2 : exp3;
- Example (similar meaning with the above statement):
  z = (a > b) ? a : b;

16

Conditional Expressions

Example:

```
int main () {
    int code, discount=0;
    code=1;
    discount = (code == 1) ? 30 : 10;
    printf(" Item discount = %d \n",discount);
    return(0);
}
```

```
int main () {
        int bil, abs;
        bil = 50;
        abs = (bil1 > 0) ? bil : - bil;
        printf("%d \n",bil);
        bil = - 50;
        abs = (bil1 > 0) ? bil : - bil;
        printf("%d \n",bil);
        return(0);
}
```


17

Logical Operator

Logical operator symbols:

Symbol	Functionality
&&	AND
||	OR
!	NOT

Logical operator truth table:

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

18

Logical Operators

Operand in Logical Operator is the operand with TRUE or FALSE value

Example :

```
int x=5; int y=0;
x && y;                  // FALSE
(x > y) && (y>=0);       // TRUE
```


19

Bitwise Operators

Symbol	Meaning	Example
&	AND	A & B
|	OR	A | B;
^	XOR	A ^ B;
~	Complement	~A;
>>	Shift Right	A >> 3;
<<	Shift Left	B << 2;

20

Bitwise Operators

BIT BY BIT OPERATION

Example:

```
int A=24; int B=35; int C;
C = A & B;   // value C = 0
C = A | B;        // value C = 59
```

Note:

A=24 binary: 011000
B=35 binary: 100011
Bit by bit AND operation resulting: 000000, in decimal: 0
Bit by bit OR operation resulting : 111011, in decimal: 59

21

Bitwise Operators

XOR operation of two bits resulting 1 if both bit are differ, and will result 0 if both are analogous.

Example:

```
int A,B=45;
A = B^75;    // Result A=102
```

Note:

Decimal 45 binary: 0101101
Decimal 75 binary: 1001011
bit by bit XOR will result in : 1100110 or 102 in decimal

22

Bitwise Operators

To create a complement-1 value, then every bit with 0 value exchange to 1 and vice versa.

Example:

```
int A, B=0xC3;
A=~B;        //value A=0x3C;
```

Note:

0xC3 binary: 1100 0011
complement-1 result:
0011 1100 in hexadecimal 3C

In C, writing a hexadecimal should start with **0x**

23

Bitwise Operators

Example :

```
int A, B=78;
A = B >> 3; //value A=9
A = B << 2; //value A=312
```

78, binary: 0100 1110
first shift right: 0010 0111
second shift right: 0001 0011
third shift right : 00001001 → 9 decimal

78, binary: 0100 1110
first shift left: 0100 11100
second shift left: 0100 111000 → 312 decimal

24

Pointer Operators

Pointer operators consist of:

& (address of)
* (value of)

will be discussed on session 13-14

25

Precedence and Associative

Every operator will have **precedence** and **associative**.

Precedence describes the order of operator execution based on its priority. Operator with the highest precedence will initially executed.

Associative describes the order of operator execution based on its location inside an expression (from left or right order). Associative will be used for operators with the same precedence level.

26

Precedence and Associative

Precedence	Operator	Associativity
1	:: (unary) ::(binary)	right to left left to right
2	() [] -> .	left to right
3	++ -- + - ! ~ (type) & sizeof	right to left
4	.* ->*	left to right
5	* / %	left to right
6	+ -	left to right
7	<< >>	left to right
8	< <= > >=	left to right
9	== !=	left to right
10	&	left to right
11	^	left to right
12	|	left to right
13	&&	left to right
14	||	left to right
15	?:	right to left
16	= += -= *= /= &= ^= |= <<= >>=	right to left
17	,	left to right

27

Precedence and Associative: Example

```
void main()
{
    float y, x;
    x = 10/3*3;
    y = 3*10/3;
    printf("x = %f\n",x); // x = 9.00
    printf("y = %f\n",y); // y = 10.00
}
```

```
#include "stdio.h"

void main()
{
    int y=10,x=10;
    y += x = 20;
    printf("x = %d\n",x); // x = 20
    printf("y = %d\n",y); // y = 30
}
```


28

References

- Paul Deitel & Harvey Deitel. (2016). C how to program : with an introduction to C++. 08. Pearson Education. Hoboken. ISBN: 9780133976892. Chapter 2, 3 & 4
- Manipulating Data with Operators: http://aelinik.free.fr/c/ch06.htm

29

END

30