Certificate - Software Development

Operators in C Language

Operators and Operands

- Various symbols we use to perform different functions are called operators.
- Typically operators are applied on values and these values are called **operands**
- Many operators take two values and they are called **binary operators**

Lelt_Operand Operator Right_Operand

Operator Classification 1

(based on Number of Operands)

1. Unary operator - !
2. Binary operator - >=
3. Ternary operator - ?:

Operator Classification 2 (on Type)

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators

Arithmetic Operators

Symbol	Operator	Example	Result
+	Addition	5 + 2	7
–	Subtraction	5 – 2	3
		-25	-25
*	Multiplication	5 * 2	10
/	Division	5 / 2	2
		5.0 / 2.0	2.5
		5 / 2.0	2.5
%	Modulus (remainder)	5 % 2	1
		15 % 100	15

- The Minus (–) operator also can be used to negate a number (as in –a + b or a + –b)
- Division (/) when applied to integers it discards any remainder and the result is a truncated integer
- But when either operand is a floating-point quantity (type float or double), the division operator yields a floating-point result
- The modulus operator % gives you the remainder when two integers are divided, and this operator can be applied only to integers

Assignment Operator

Syntax:

***variable = value*;**

variable = expression;

- The assignment operator (=) assigns a value to a variable. If right hand side is an expression, the result is assigned to the variable.

```
num = 1;
```
// sets num to 1

```
num = temp;
```
// sets num to value of temp

```
num = a + b;
```
// result of r.h.s. expression stored in num

- C Language allows chain assignment

```
num = temp = a
```
// right most values get assigned to all variables

Relational Operators

The relational (comparison) operators take two values and return a value of 1 or 0 depending on whether the comparison was true or false. The complete set of relational operators in C are:

Symbol	Operator	Example	Result
<	Less than	5 < 10	1
<=	Less than or equal	6 <= 3	0
>	Greater than	5.3 > 10	0
>=	Greater than or equal	'B' >= 'A'	1
==	Equal	5 + 2 == 7	1
!=	Not equal	'A' != 65	0

Note :-

- Equal operator is **==** sign (double equals) as the = sign the assignment operator in C.
- Single characters also can be compared with integers as the compiler substitute ASCII values for them.
- Strings cannot be compared with there operators "hello" <= " house" **X**

Logical Operators

You can also combine true/false values by using the Logical operators, which take true/false values as operands and compute new true/false values. The three Logical operators are:

Symbol	Operator	Example	Result
&&	Logical AND	5 > 10 && 5 <= 10	0
||	Logical OR	5 > 10 || 5 <= 10	1
!	NOT operator	! (5 <=10)	0

Note :-

- Reason for using ( ) with ! operator is, that it has higher **precedence** than relational operators. We use the ( ) to evaluate the condition first and then invert it.

++ and – – Operators

- ++ (Increment Operator) – which increases the value of a variable by 1

```
++num  or  num++  is equivalent to  num = num + 1
```

- – – (Decrement Operator) - which decreases the value of a variable by 1

```
--num  or  num--  is equivalent to  num = num -1
```

- **Pre** Increment / decrement – **operator variable** (e.g., ++num or −−num)

  When the operator is applied before the variable the value of the variable is first changed and then the **new value** is used in the expression.

- **Post** Increment / decrement – **variable operator** (e.g., num++ or num−−)

  When operators is applied after the variable the **current value** of the variable is used in the expression and then the variable is changed.

Pre and Post Application - Example

Pre and Post Example

```
main()
{
   int temp, n=5, res = ++n;
   printf("PRE : n = %d, res = %d\n", n, res);
   n=5; res = n++;
   printf("POST: n = %d, res = %d\n", n, res);
   n=4; temp = --n / 2;
   printf("PRE : n = %d, temp = %d\n", n, temp);
   n=4; temp = n-- / 2;
   printf("PRE : n = %d, temp = %d\n", n, temp);
   char c='A', new = c++;
   printf("POST: c = %c, newc = %c\n", c, newc);
}
```

```
PRE : n = 6, res = 6
POST: n = 6, res = 5
PRE : n = 3, temp = 1
POST: n = 3, temp = 2
POST: c = B, newc = A
```

Bitwise Operators

Op	Example	Example	Results
~	Bitwise Negation	~ 00000111	11111000
&	Bitwise AND	00000111 & 00000101	00000101
|	Bitwise OR	00000111 | 00000101	00000111
^	Bitwise XOR	00000111 ^ 00000101	00000010
>>	Right shift	00000111>>2	00000001
<<	Left shift	00000111<<1	00001110

Bitwise Operator - Example

```
main()
{
   char x=7, y=5;
   printf("~x  = %d\n", ~x);
   printf("x&y  = %d\n", x&y);
   printf("x|y  = %d\n", x|y);
   printf("x^y  = %d\n", x^y);
   printf("x>>2 = %d\n", x>>2);
   printf("x<<1 = %d\n", x<<1);
}
```

Output
~x = -8
x&y = 5
x|y = 7
x^y = 3
x>>2 = 1
x<<1 = 14

Assignment Operators

Operator	Example	Equivalent
+=	tot += num	tot = tot + num;
-=	pay -= tax	pay = pay – tax;
*=	fact *= num;	fact = fact * num;
/=	avg /= num;	avg = avg / num;
%=	rem %= 2	rem = rem % 2

Operator Precedence

Operator Set	Description
() [] . ->	Parenthesis, Array subscript, Structure member
++ --	Increment, Decrement
! ~	Not Operator, Negation
* / %	Multiplication, Division, Modulus
+ –	Addition, Subtraction
>> <<	Shift Operators
< <= > >= == !=	Relational Operators
& | ^	Bitwise Operators
&& ||	Logical Operators
?:	Conditional Operator
= += -= *= /= %=	Assignment Operators

Lesson Summary

- Operators and Operands
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Increment / Decrement Operator
- Pre and Post Application
- Assignment Operators
- Bitwise Operators
- Operator Precedence