Msc-Bioinformatics(Sem- 1)

COMPUTER FUNDAMENTALS, NETWORKING ‘WEB TECHNOLOGY AND BASICS OF C 3 75 PROGRAMMING LANGUAGE

Operators & Its Types

By: Nishi
Assist. Prof. in Comp. sc. & IT

INTRODUCTION TO OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

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 Following table shows all the arithmetic operators supported by C language. Assume variable A holds 10 and variable B holds 20 then:

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

a+b = 13 a-b = 5

Example: Arithmetic Operators

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a = 9,b = 4, c;
c = a+b;
printf("a+b = %d \n",c);
c = a-b;
printf("a-b = %d \n",c);
c = a*b;
printf("a*b = %d \n",c);
c = a/b;
printf("a/b = %d \n",c);
c = a%b;
printf("Remainder when a divided by b = %d \n",c);
return 0;
}
```

Output

```
a+b =13
a-b= 5
a*b=36
a/b=2
Remainder when divided
by b= 1
```

Increment and Decrement Operators

- C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.
- Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Example 2: Increment and Decrement Operators

```
#include<stdio.h>
#include<conio.h>
{
int main()
{
int a = 10, b = 100;
float c = 10.5, d = 100.5;
printf("++a = %d \n", ++a);
printf("--b = %d \n", --b);
printf("++c = %f \n", ++c);
printf("--d = %f \n", --d);
return 0;
}
```

Output

```
++a = 11
--b = 99
 ++c = 11.500000
 ++d = 99.500000
```

Assignment Operators

- An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	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

c = 5 c = 10 c = 5 c = 25 c = 5 c = 0

Example : Assignment Operators

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a = 5, c;
c = a;
printf("c = %d\n", c);
c += a;
printf("c = %d\n", c);
c -= a;
printf("c = %d\n", c);
c *= a;
printf("c = %d\n", c);
c /= a;
printf("c = %d\n", c);
c %= a;
printf("c = %d\n", c);
return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

C Relational Operators

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Example : Relational Operators

```
#include <stdio.h>
#include<conio.h>
int main()
{
int a = 5, b = 5, c = 10;
printf("%d == %d is %d \n", a, b, a == b);
printf("%d == %d is %d \n", a, c, a == c);
printf("%d > %d is %d \n", a, b, a > b);
printf("%d > %d is %d \n", a, c, a > c);
printf("%d < %d is %d \n", a, b, a < b);
printf("%d < %d is %d \n", a, c, a < c);
printf("%d != %d is %d \n", a, b, a != b);
printf("%d != %d is %d \n", a, c, a != c);
printf("%d >= %d is %d \n", a, b, a >= b);
printf("%d >= %d is %d \n", a, c, a >= c);
printf("%d <= %d is %d \n", a, b, a <= b);
printf("%d <= %d is %d \n", a, c, a <= c);
return 0;
}
```

OUTPUT

```
5 == 5 is 1
5 == 10 is 0
 5 > 5 is 0 5 > 10 is 0
 5 < 5 is 0
 5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
 5 >= 5 is 1
 5 >= 10 is 0
5 <= 5 is 1
 5 <= 10 is 1
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.

Logical operators are commonly used in decision making in C programming.

Logical Operators

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

EXAMPLE: LOGICAL OPERATORS

```
#include <stdio.h>
#include<conio.h>
int main()
{
int a = 5, b = 5, c = 10, result;
result = (a == b) && (c > b);
printf("(a == b) && (c > b) is %d \n", result);
result = (a == b) && (c < b);
printf("(a == b) && (c < b) is %d \n", result);
result = (a == b) || (c < b);
printf("(a == b) || (c < b) is %d \n", result);
result = (a != b) || (c < b);
printf("(a != b) || (c < b) is %d \n", result);
result = !(a != b);
printf("!(a == b) is %d \n", result);
result = !(a == b);
printf("!(a == b) is %d \n", result);
return 0;
}
```

OUTPUT

```
(a == b) && (c > b) is 1
 (a == b) && (c < b) is 0
(a == b) || (c < b) is 1
 (a != b) || (c < b) is 0
 !(a != b) is 1 !(a == b) is 0
```

Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

BITWISE OPERATORS

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.
Let us suppose the bitwise AND operation of two integers 12 and 25.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bit Operation of 12 and 25

```
  00001100
& 00011001
----------
  00001000 = 8 (In decimal)
```

Bitwise AND

```
#include <stdio.h>
#include<conio.h>
int main()
{
int a = 12, b = 25;
printf("Output = %d",
a&b);
return 0;
}
```

OUTPUT

OUTPUT=8

Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

00001100

| 00011001

_____

00011101 = 29 (In decimal)

Bitwise OR operator |

```
#include <stdio.h>
#Include<conio.h>
int main()
{
int a = 12, b = 25;
return 0;
printf("Output = %d", a|b);
}
```

OUTPUT

OUTPUT=25

BITWISE XOR(EXCLUSIVE OR)

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

00001100
^ 00011001

_____

00010101 = 21 (In decimal)

BITWISE XOR(EXCLUSIVE OR)

```
#include <stdio.h>
#include<conio.h>
int main()
{
    int a = 12, b = 25;
printf("Output = %d", a^b);
return 0;
}
```

OUTPUT

OUTPUT=21

Bitwise compliment operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

35 = 00100011 (In Binary)

Bitwise complement Operation of 35

~ 00100011

_____

11011100 = 220 (In decimal)

2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1.

Decimal	Binary	2's complement
0	00000000	-(11111111+1) = -00000000 = -0(decimal)
1	00000001	1 -(11111110+1) = -11111111 = -256(decimal)
12	00001100	-(11110011+1) = -11110100 = -244(decimal)
220	11011100	-(00100011+1) = -00100100 = -36(decimal)

Shift Operators in C programming

There are two shift operators in C programming:

- Right shift operator
- Left shift operator.

Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)

Left Shift Operator

Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by <<.

212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 =11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)

Left Shift Operator

```
#include <stdio.h>
#nclude<conio.h>
int main()
{
int num=212, i;
for (i=0; i<=2; ++i)
printf("Right shift by %d: %d\n", i,
num>>i);
      printf("\n");
for (i=0; i<=2; ++i)
printf("Left shift by %d: %d\n", i,
num<<i);
return 0;
}
```

```
Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848
```

Comma Operator

Comma operators are used to link related expressions together.

For example:

1.int a, c = 5, d;
2. float sum,sub,total;

SIZE OPERATOR

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

```
#include <stdio.h>
#include<conio.>
int main()
{
int a;
float b;
char d;
printf("Size of int=%lu bytes\n",sizeof(a));
printf("Size of float=%lu bytes\n",sizeof(b));
printf("Size of double=%lu bytes\n",sizeof(c));
printf("Size of char=%lu byte\n",sizeof(d));
return 0;
double c;
}
```

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.

Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

Operators Precedence

For example, x = 7 + 3 * 2
here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Operators Precedence

Category	Operator	
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right

Operators Precedence

Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

THANK YOU