PROFESSIONAL TRAINING INSTITUTE

Assignment - IV

An ISO9001:2015 certified training institute

Assigment IV - **Operators**

C language supports a rich set of built-in operators. 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

1.1 Arithmetic operators:

C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

1.2 Relational operators:

The following table shows all relational operators supported by C.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

1.3 Logical operators:

C language supports following 3 logical operators. Suppose a = 1 and b = 0,

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

1.4 Bitwise operators:

Bitwise operators perform manipulations of data at **bit level**. These operators also perform **shifting of bits** from right to left. Bitwise operators are not applied to float or double(These are data types, we will learn about them in the next tutorial).

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Now let's see truth table for bitwise &, | and ^

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise **shift** operator, shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value have to be shifted. Both operands have the same precedence.

1.5 Assignment Operators:

Assignment operators supported by C language are as follows.

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

1.6 Conditional operator:

The conditional operators in C language are known by two more names

1. **Ternary Operator**
2. **? : Operator**

It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is:

Explanation:

- The question mark **"?"** in the syntax represents the**if** part.
- The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
- If (expression 1) returns true then the expression on the left side of **" : "**i.e (expression 2) is executed.
- If (expression 1) returns false then the expression on the right side of **" : "**i.e (expression 3) is executed.

1.7 Special operator:

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x** ; return address of the variable **x**
*	Pointer to a variable	***x** ; will be pointer to a variable **x**

Precedence and associativity:

Precedence and associativity of C operator affect grouping and evaluation of operation in expression. Is meaningful only if other operators having a higher or lower precedence is present. Expression with higher precedence operator only operate first.The precedence and associativity (the order in which the operands are evaluated) of C operators.In the order of precedence from highest to lowest.If several operators appear together, they have equal precedence and are evaluated according to their associativity.

All simple and compound-assignment operators have equal precedence. Operators with equal precedence such as + and -, evaluation proceeds according to the associativity of the operator, either from right to left or from left to right.

A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro

You create macros with the '#define' directive. '#define' is followed by the name of the macro and then the token sequence it should be an abbreviation for, which is variously referred to as the macro's body, expansion or replacement list .

For example,#define BUFFER_SIZE 1024

Defines a macro named BUFFER_SIZE as an abbreviation for the token 1024. Defines a macro named BUFFER_SIZE as an abbreviation for the token 1024. If somewhere after this '#define' directive there comes a C statement of the form.
foo = (char *) malloc (BUFFER_SIZE);
Then the C preprocessor will recognize and expand the macro BUFFER_SIZE. The C compiler will see the same tokens as it would if you had writtenfoo = (char *) malloc (1024);

By convention, macro names are written in uppercase. Programs are easier to read when it is possible to tell at a glance which names are macros.
When the preprocessor expands a macro name, the macro's expansion replaces the macro invocation, then the expansion is examined for more macros to expand.

For example,
#define TABLESIZE BUFSIZE#define BUFSIZE
1024TABLESIZE 7
→ BUFSIZE 7
→ 1024
TABLESIZE is expanded first to produce BUFSIZE, then that macro is expanded to produce the final result, 1024. Notice that BUFSIZE was not defined when TABLESIZE was defined. The '#define' for TABLESIZE uses exactly the expansion you specify—in this case, BUFSIZE —and does not check to see whether it too contains macro names. Only when you use TABLESIZE is the result of its expansion scanned for more macro names.

This makes a difference if you change the definition of BUFSIZE at some point in the source file. TABLESIZE, defined as shown, will always expand using the definition of BUFSIZE
That is currently in effect:
#define BUFSIZE 1020
#define TABLESIZE BUFSIZE
#undef BUFSIZE
#define BUFSIZE 37
As an example, here is a macro that computes the minimum of two numeric values, as it is defined in many C programs, and some uses.
#define min(X, Y) ((X) < (Y) ? (X) : (Y))
x = min(a, b); → x = ((a) < (b) ? (a) : (b));
y = min(1, 2); → y = ((1) < (2) ? (1) : (2));
z = min(a + 28, *p); → z = ((a + 28) < (*p) ? (a + 28) : (*p));
Exercises:

1. Assignment
   a. Write macro using conditional operator for finding maximum number (a:b ?a:b.).
   b. Write a program to right shift a variable (0x45Fe6A92) by two times and print it.
   c. Write a program to left shift a variable (0xA) by one, two, three, four times and print and analyze.
   d. Write a program to right shift a variable (0xA00) by one, two, three, four times and print and analyze.
   e. Take two numbers and perform the bit wise AND operation.
   f. Take two numbers and perform the bit wise OR operation.
   g. Take a number in variable and perform compliment operation.
   h. g = a / 2 + a * 4 / a - a + i/ 3 ; (i = 2.5, a = 2).
   i. o= i * a / 2 + 3 / 2 * a + 2 + t ; (i = 4, a = 1, t = 3).
   j. a=2,b=7,c=10; By using given value find the value of c in the given equation c=a==b.
   k. s = q * a / 4 - 6 / 2 + 2 / 3 * 6 / g; (q = 4, a= 2, g = 2).
   l. What is the final value of a a=015 + 0x71 +5 .
   m. s = 1 / 3 * a / 4 - 6 / 2 + 2 / 3 * 6 / g ; (a = 4, g = 3).
   n. i=5, j=++i+++i+++i What is the output of j?.
   o. By using bitwise operator ((~7 & 0x000f) ==8), Explain the operation of this equation?
   p. Find the value of Z, z = y && (y |= 10); assume the Y value is 0.?
   q. a=a++ + ~++a; a=2?
   r. Check the entered character is vowel or consonant
   s. Using bitwise operator find given number is even or odd.
   t. Repeat questions 2,5,7,19 with macro.

2. Theory question need to clear

   a) What is precedence and associativity?
   b) What is ternary operator?
   c) How to multiple number with 4 without using * operator?
   d) How to divide number with 32 without using / operator?
   e) Which bitwise operator is suitable for checking whether a particular bit is on or off?
   f) & operator | which operator will have high priority?
   g) In the given number 0xFC45DA3A shift this by 6 bit right side?
   h) Which operator we can use for bit extraction & or |
   i) What is post increment and pre incrementing operator?

Check as many questions as possible for operator on internet, as many question will come in written test from operator.