C - Programming

Operators and Expressions

Jayakumar P

Department of Computer Science and Applications

Operators in C

- Operators are symbols used to manipulate data. Data may be in the form of variables, constants, or values returned by functions.

- For example,
  a + b *(a, b are variables/constants)*
  2 + 3 *(2, 3 are values)*
  a + pow(2,3) *(value returned by pow function is added with a)*

- **Categories of operators**
  1. Arithmetic Operators
  2. Logical operators
  3. Relational operators
  4. Bitwise operators
  5. Assignment operator
  6. Other (Miscellaneous) operators

Arithmetic Operators

- There are 5 arithmetic operators in C
    1. Addition (+)
    2. Subtraction (-)
    3. Multiplication (*)
    4. Division (/)
    5. Modulus (%)
- Addition, Subtraction, Multiplication, and Division can be applied on any numeric data type
- Modulus can be applied only on integer data.

Integer and Float division

- If any one (or both) of the operands is float, then the division operator performs ordinary division.
  For example, 2.5/1 → 2.5

- If both operands are integers, then division operator performs integer division and drops the fractional part of the result. See the program.

- Typecast numerator or denominator (to float) to avoid loss of precision.
  For example, the following statement gives correct answer.
  printf("%f", c/(float)a);

```
 9  #include<stdio.h>
10  int main()
11  {
12      int c;
13      int a;
14      scanf("%d%d",&c,&a);
15      printf("%f",c/a);
16      return 0;
17  }
18
```

```
main.c:15:11: warning: format '%f' expects a
2
3
0.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

Logical Operator

- Logical operators are applied on (True/False) values.
- In C language, True and False are represented as follows.
  - True $\rightarrow$ any value other than 0
  - False $\rightarrow$ 0
- Logical operators returns 1 if the expression evaluates to a True and 0 otherwise.
- The three logical operators are Logical AND (&&), Logical OR (||), and Logical NOT (!)

p	q	p&&q	p||q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Relational Operator

- There are six relational operators in C
- All relational operators are binary operators and both of the operands must be numeric or character values.
- The result of relation operators is either a 1 (representing true) or 0 (representing false).

Operator	Name
==	Equal
!=	Not Equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

Bitwise Operator

- There are six bitwise operators in C
- Bitwise operators are used to manipulate individual bits in the binary representation of numeric data

Operator	Name
&	Bitwise AND
|	Bitwise OR
«	Left Shift
»	Right Shift
~	Bitwise NOT
∧	Bitwise XOR

x	00001001
y	00010111
x&y	00000001
x|y	00011111
x∧y	00011110
~ x	11110110
x«2	00100100
x»2	00000010

As an example, Let x = 9, binary of x is 00001001, y = 23, binary of y is 00010111

Assignment Operators

- **Simple Assignment operator** (=) is used to assign (store) the value of an expression in an identifier (variable). General form is as follows
  identifier = expression
- For example,
  a = 3; Stores the integer value 3 in the variable a.
  x = 2+5; This statement first adds 2 and 5, and stores the result (7) in the variable x.
- If two operands in an assignment statement are of different data types, then the type of the left operand dominates. For example, The following assignment statement stores 3 in the variable x.
  int x;
  x = 3.5;

Compound Assignment operators

- Compound assignment operators first uses the existing value in a variable and then stores the result in the same variable.
- Following are some of the compound assignment operators in C. Assume that initial value of x = 2;

Operator	Example	Explanation
+=	x += 3;	x = x + 3
-=	x -= 3;	x = x - 3
*=	x *= 3;	x = x * 3
/=	x /= 3;	x = x / 3
%=	x %= 3;	x = x % 3

Miscellaneous Operators

- Conditional Operator (?:)- Also called ternary operator in C
  expression 1 ? expression 2 : expression 3
  - Initially expression 1 is treated as the condition and if it evaluates to a true, then expression 2 is evaluated otherwise expression 3 is evaluated.
  - For example the following conditional operator finds the smallest among two numbers. Let x =1, y= 2,
    min = x<y? x:y;
- Comma Operator
  - Used to evaluate multiple expressions, separated by comma - from left to right order.
  - The result of the right most expression is the result of comma operator. For example the expression **j = (1,2,3)** stores the value 3 in the variable j.
- sizeof() Operator
  Used to calculate the size of any data item or type.
  For example **sizeof(int)** gives 4.

Unary Operators

- Applied on a single operand
- **Unary Increment and Unary Decrement**.

Operator	Name	Description
x++	Post Increment	x is used first and increment is done post usage.
++x	Pre Increment	x is first incremented and then used
x- -	Post Decrement	x is used first and decrement is done post usage.
- -x	Pre Increment	x is first decremented and then used

Precedence and Associativity of Operators

- Precedence and Associativity of operators are properties used to evaluate expression consisting of multiple operators
- **Precedence** - How to evaluate an expression consisting of multiple operators?
  Operators of higher precedence is applied before the operators of lower precedence in the expression
  Evaluate **x = 3 + 4 / 2**. Solution. Apply / first, then + and finally =.
- **Associativity** - How to evaluate an expression consisting of multiple operators with same precedence? - Based on the associativity of operators.
  Evaluate **x = 4 / 2 * 3** Solution. Apply / first, then * and finally = .
  Because the operators / and *, are left to right associative.

The precedence table

- The precedence table - where operators are listed in decreasing order of precedence. All operators in same line has same precedence.

TABLE 2-1. PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

OPERATORS	ASSOCIATIVITY
`() [] -> .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

Q1

$a = 3 + 9/2$

Q1-Solution

1. The sub-expression $9/2$ is evaluated first (result 4.5) and the resultant expression is $a = 3 + 4.5$
2. The expression $3 + 4.5$ is evaluated (result 7.5) and the resultant expression is $a = 7.5$
3. $a = 7.5$ is evaluated (the value 7.5 is assigned to the variable a).

Q2

$$a = (3 + 9)/2$$

Q2-Solution

$a = (3 + 9)/2$

1. (3+9) — Precedence.
   Resultant expression $a = 12/2$
2. $12/2 \rightarrow 8$ — Precedence
   Resultant expression $a = 6$
3. $a = 6$

Q3

$$a = 7 * 6\%5/3$$

Q3-Solution

$a = 7 * 6\%5/2$

The three operators *, %, and / have same precedence. So associativity rules apply. All the operators are left to right associative. So they must be evaluated in left to right order.

1. 7*6
   Resultant expression is $a = 42\%5/2$
2. 42%5
   Resultant expression is $a = 2/2$
3. 2/2
   Resultant expression is $a = 1$
4. $a = 1$

Q4

a = 7 * ((6 % 10)/3)

Q4-Solution

a = 7 * ((6 % 10)/3)
The innermost parenthesis is evaluated first. Then the outer parenthesis.

1. 6%10
   Resultant expression is $a = 7 * (6/3)$
2. 6/3
   Resultant expression is $a = 7 * 2$
3. $7 * 2$
   Resultant expression is $a = 14$
4. $a = 14$

Q5

```
a = (b = 3 * 4) / 5;
```

Q5-Solution

a = (b = 3 * 4) / 5;
The parenthesised sub-expression is evaluated first. It has two operators, = and *. Out of these two operators * has higher precedence, so the sub expression 3*4 is evaluated first.

1. 3 * 4
   Resultant expression is $a = (b = 12)/5$
2. b = 12
   Resultant expression is $a = b/5$
3. $b/5$
   Resultant expression is $a = 2$
4. $a = 2$

Q6

```
a = -15 * 2 + 3;
```

Q6-Solution

a = -15 * 2 + 3;

1. -15
   Unary (-) has higher precedence
2. (-15)*2
   Resultant expression is $a = (-30) + 3$
3. $(-30) + 3$
   Resultant expression is $a = (-27)$
4. $a = -27$

Q7

Let x = 2 and y= 3. What is the values of the variables x, y and a after the evaluation of the following expression?

$a = --x + y++$

Q7-Solution

Let x = 2 and y= 3

$a = --x + y++$

1. $--x$ is evaluated first (unary pre-decrement)
   Resultant expression $a = 1 + y++$
2. 1+y is evaluated and then value of y is incremented (since y++ is post increment)
   Resultant expression $a = 1 + 3, y = y + 1$
3. $a = 4$

Output

$x = 1$

$y = 4$

$a = 4$

Q8

A frog is currently at the point 0 on the coordinate line. It jumps towards right and left as follows. At first, it jumps a units to the right, then the second jump is b units to the left, the third jump is again a units to the right, then the fourth jump is b units to the left, and so on. Calculate the position of the frog after k jumps.

Q8-Solution

What are the inputs? a,b and k
If k is even, the frog makes $k/2$ jumps to right and left each. So final position is $(k/2) * a - (k/2) * b$.
If k is odd, then the frog makes $k/2 + 1$ jumps to right and $k/2$ jumps to left . So, final position is $(k/2 + 1) * a - (k/2) * b$
Write the program.

Q9

Two friends are on the coordinate line, at points with integer coordinates. One of them is at the point a and the other is at the point, b. They can move along the line in any direction unlimited number of times. When a friend moves his tiredness changes according to the following rules:

The first move increases the tiredness by 1, the second move increases the tiredness by 2, the third by 3 and so on. For example, if a friend moves first to the left, then to the right (returning to the same point), and then again to the left his tiredness becomes equal to 1+2+3=6.

The friends want to meet at a integer point with minimum total tiredness. Find the total minimum tiredness they will have.

Q9-Solution

If one friend makes all the moves (and the other stands still) then the total tiredness will be maximum. So both of them should make nearly equal number of steps.

If d is the total distance between them, friend 1 should make $floor(d/2)$ steps and friend 2 should make $ceil(d/2)$ steps.