C Language Basics (Part 2)

Outline

- 1. Operators
- 2. Arithmetic Operators
- 3. Operator Precedence and Associativity
- 4. Expressions
- 5. Different Forms of Assignment Operators
- 6. Increment and Decrement Operators
- 7. Swapping Values between Two Variables

1. Operators

Operator – a symbol or keyword that represents an operation to be applied to some data in the program

```
e.g.: varA = -varB + 40 * 20 ;
```

- Operand input to an operator
- Binary operator an operator that accepts 2 operands
 e.g.: 40 * 20
- Unary operator an operator that accepts 1 operand

```
e.g.: -varB
```

How about ternary operator? (a > b) ? 1 : 0

2. Arithmetic Operators

Operator	Description	Example
+	Addition	8 + 5 → 13
-	Subtraction	8 - 5 → 3
*	Multiplication	8 * 5 > 40
/	Division	8 / 5 \rightarrow 1 (Note: <i>Integer division</i>) 8.0 / 5.0 \rightarrow 1.6
%	Modulus (yields the remainder of a division)	8 % 5 → 3 What happen for 8 / 0?
	Note : Applicable only to integers	How about 8 % 0?

2. Arithmetic Operators

 When used <u>as an unary operator</u>, '-' becomes a negation operator, which turns positive value into negative value and vice versa.

```
e.g.: foo = 5 ;
bar = -foo; // Assign -5 to bar
```

Exercise: evaluate the following expressions

- 20 % 3
- 2 % 9
- 30 / 20 / 2
- 10 * 2 + 4 * 3

Some uses of Integer Division and Modulus Operators

Suppose **n** is an integer

- (n % 10) yields the right most digit of n
 e.g.: 1234 % 10 → 4
- (n / 100 % 10) yields the 3rd digit from the right of **n** e.g.: 1234 / 100 % 10 → 12 % 10 → 2
- Determining if n is odd or even
 if n is even, (n % 2) is 0
 if n is odd, (n % 2) is 1 or -1 (i.e., not zero)

3. Operator Precedence & Associativity

How should we evaluate the following expression?
 In what order should the operators be applied?

- Among different operators, operator precedence tells us which operator(s) should be applied first.
- Among operators with the same precedence,
 operator associativity tells us whether the left-most
 or the right-most operator should be applied first.

3. Operator Precedence & Associativity

Operators	Associativity	Precedence
(postfix) ++ (postfix)	left to right	Highest
+ (unary) - (unary) ++ (prefix) (prefix)	right to left	^
* / %	left to right	
+ -	left to right	
= += -= *= /= etc.	right to left	Lowest

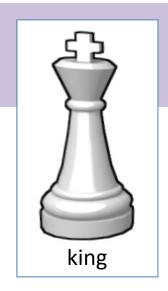
• Operators at the same level have the same precedence.

• 2 - 25 /
$$10 + 33 \% 10 * 2 = ?$$

3.1. Parentheses

 Use parentheses '('and ')' to explicitly specify the evaluation order of sub-expressions

$$(a + b) * (c + d)$$



Multiple levels of parentheses (Cannot use [] or { })

$$((a + b) * (a + b) - c) * (d - e)$$

• **Tips:** Use parentheses <u>for clarity</u> or when you are not sure about the precedence of the operators.

4. Expressions

 An expression is a combination of operators, constants, variables, and function calls

```
- e.g.: 30
24 + a
d = b * b - 4 * a * c
sqrt( 4.0 ) + a * sqrt( 9.0 )
```

- An expression
 - Can always be evaluated to a value (of some data type)
 - Can be part of another expression

Note: **sqrt** is a math function for computing the square root

5. Assignment Operators

variable = expression

- Low precedence, *right-to-left* associativity
- expression is evaluated first and the evaluated value is assigned to variable.
- "variable = expression" is also an expression, which evaluates to the value of variable.

```
e.g.:
```

$$var1 = var2 = 3 + 2$$

is evaluated as

$$var1 = (var2 = (3 + 2))$$

This may not work on other programming languages, e.g., Python

```
1 int a = 0 , b = 2 , c ;
2 double pi = 3.1416 ;
3
```

Assignment operator can be used to initialize variables in variable declaration.

```
Equivalent to
int a , b , c ;
double pi ;
a = 0 ;
b = 2 ;
pi = 3.1416 ;
```

What's the value of variable **c**?

```
1  int a = 0;
2  
3  a = a + 2;
4  printf( "%d" , a );  // What's the output?
```

+ has higher precedence than =. Thus

$$a = a + 2$$

is evaluated as

$$a = (a + 2) \rightarrow a = (0 + 2) \rightarrow a = 2$$

```
1  int a = 1 , b = 2 ;
2  b = b * a ;
3  a = 0 ;
4  printf( "%d" , b );  // What's the output?
```

Statements are executed sequentially one after another.

```
(Line 1) a is set to 1 and b is set to 2.
```

(Line 2) b becomes 2.

(Line 3) a becomes 0 but changing a does not affect other variables.

```
int b , c , d ;
d = c = b = 0 ;  // Assign 0 to variables b, c, and d
// d = c = b = 0 is evaluated as d = (c = (b = 0))
```

5.1. Assignment Operators – Short Form

- $\mathbf{i} = \mathbf{i} + \mathbf{2}$; can be written as $\mathbf{i} += \mathbf{2}$;
- The semantics of
 variable = variable op (expression);
 is equivalent to
 - variable op= expression;
- Some short form assignment operators:

• Note: $\mathbf{i} *= \mathbf{j} + \mathbf{2}$; is equivalent to $\mathbf{i} = \mathbf{i} * (\mathbf{j} + \mathbf{2})$; and not to $\mathbf{i} = \mathbf{i} * \mathbf{j} + \mathbf{2}$;

6. Increment / Decrement Operator

• To increase the value of a variable, **i**, by one, we can write the following statement:

$$i = i + 1;$$

 In C language, we can write a statement with an increment operator to achieve the same result like this:

$$i++;$$
 or $++i;$

• Similarly, we can write **i--** or **--i** to decrease the value of **i** by one.

 The increment operator (++) can be placed in either prefix or postfix position, with different results.

• ++i (prefix)

- Increase the value of \mathbf{i} by 1
- The <u>value of the expression</u> "++i" is the value of i after the increment operation.

• i++ (postfix)

- The <u>value of the expression</u> "i++" is the value of i before the increment operation.
- Increase the value of \mathbf{i} by 1

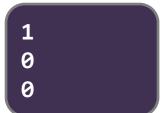
Statement that involves ++ operator	Equivalent statements
<pre>j = ++i ; // prefix increment</pre>	<pre>i = i + 1; // side effect first j = i;</pre>
<pre>j = i++; // postfix increment</pre>	<pre>j = i ; i = i + 1 ; // side effect last</pre>
<pre>printf("%d\n" , ++j); // prefix increment</pre>	<pre>j = j + 1; // side effect first printf("%d\n" , j);</pre>
<pre>printf("%d\n" , j++);</pre>	<pre>printf("%d\n" , j); j = j + 1 ; // side effect last</pre>

Example

```
1 int i , j ;
2 i = 0 ;
3 j = ++i ;
4 printf( "%d\n" , i );
5 printf( "%d\n" , j );
6 printf( "%d\n" , ++j );
```

```
1
1
2
```

```
1 int i , j ;
2 i = 0 ;
3 j = i++ ;
4 printf( "%d\n" , i );
5 printf( "%d\n" , j );
6 printf( "%d\n" , j++ );
```



Try to avoid doing nasty things; you don't need to

```
int i = 0 , j = 0 ;
i = i++    ; printf( "ij = %d,%d\n" , i , j );
i = ++i    ; printf( "ij = %d,%d\n" , i , j );
i = i+++i    ; printf( "ij = %d,%d\n" , i , j );
i = i+++j    ; printf( "ij = %d,%d\n" , i , j );
i = i+++i    ; printf( "ij = %d,%d\n" , i , j );
i = i---i    ; printf( "ij = %d,%d\n" , i , j );
i = i----i    ; printf( "ij = %d,%d\n" , i , j );
i = i+--+-i    ; printf( "ij = %d,%d\n" , i , j );
i = i++-++i    ; printf( "ij = %d,%d\n" , i , j );
```

```
How about... i = ++i++ ;
i = i++++i ;
i = i++++i ;
```

7. Swap the value between two variables

```
int a = 0 , b = 1 , tmp ;
// How to exchange/swap the value of a and b?
           // Method A ?
  a = b;
   b = a;
  tmp = b;
                       // Method B ?
   b = a;
   a = tmp;
                       // Method C ?
  tmp = b;
  a = tmp;
                             Is it possible to swap variables
                                without using "tmp"?
Answer: Method B
                                Hint: a = a + b; // then?
```

Summary

- Arithmetic operators (+, -, *, /, %)
- Operator precedence and associativity
- Different forms of assignment operators (=, +=, -=, *=, ...)
- Increment (++) and decrement (--) operator
- Swap the value between two variables

Next: Data Types