Tutorial Link https://codequotient.com/tutorials/C - Precedence and Associativity of operators/5a1d9c1252795c1b16c0ac26

TUTORIAL

C - Precedence and Associativity of operators

Chapter

1. C - Precedence and Associativity of operators

If there are more than one operators in a single expression, then a ordering must be done between these operators, for which will execute in which sequence. There are two rules specified known as operator precedence and operator associativity.

When the operators are of different type the precedence will lead the ordering between operators.

But When the operators are of same type the associativity will lead the ordering between operators.

Following is the precedence and associativity rules in C languages:

Precedence Operators   Associativity

```
Highest      () [] . →                                    Left to
right   (Parentheses etc.)
             ! ~ ++ -- sizeof + - * & (type)              Right to
left   (Unary operators)
             * / %                                        Left to
right   (Arithmetic)
             +  -                                         Left to
right   (Arithmetic)
             >> <<                                        Left to
```

```
right   (Bitwise Shift)
             < <= > >=                                    Left to
right   (Relational)
             == !=                                        Left to
right   (Relational)
             &                                            Left to
right   (Bitwise AND)
             ^                                            Left to
right   (Bitwise XOR)
             |                                            Left to
right   (Bitwise OR)
             &&                                           Left to
right   (Logical AND)
             ||                                           Left to
right   (Logical OR)
             ?:                                           Right to
left   (Conditional)
             = += -+ *= /+ *= %= &= ^= |= >>=             Right to
left   (Assignment)
Lowest       ,                                            Left to
right   (Comma)
```

```c
#include <stdio.h>

int main()
{
  int a,b,c,d;
  a = 15;
  b = 2;
  c = 5;
  d = 8;

  printf("a=%d b=%d c=%d d=%d \n\n",a,b,c,d);

  printf("a * b - c = %d \n", a * b - c);
  printf("a - b * c = %d \n", a - b * c);

  return 0;
}
```

C does not specify the order in which the subexpressions of an expression are evaluated. This leaves the compiler free to rearrange an expression to produce more optimal code. However, it also means that your code should never rely upon the order in which subexpressions are evaluated. For example, the expression

```
x = f1() + f2();
```

does not ensure that f1( ) will be called before f2( ). f1() may be evaluated before f2() or vice versa; thus if either f1() or f2() alters a variable on which the other depends, x can depend on the order of evaluation.

Similarly, the order in which function arguments are evaluated is not specified, so the statement

```
printf("%d %d\n", ++n, power(2, n));
```

can produce different results with different compilers, depending on whether n is incremented before power is called. The solution, of course, is to write

```
++n;
printf("%d %d\n", n, power(2, n));
```

According to Dennis Ritchie - “The moral is that writing code that depends on order of evaluation is a bad programming practice in any language. Naturally, it is necessary to know what things to avoid, but if you don't know how they are done on various machines, you won't be tempted to take advantage of a particular implementation”.