Operator Precedence Lab

Introduction

Order of operations (also known as operator precedence) is a set of rules used to clarify the order in which procedures are performed in mathematical expressions. Similarly, operator precedence exists within computer programming and varies from programming language to programming language.

Disregard or confusion about operator precedence has been the cause of many software malfunctions. In these cases, the mathematical operators are performed in an order contrary to what the software developer predicted and thus return an incorrect value. These errors can cause anything from the loss of hundreds of millions of dollars to forced rocket explosions.

Review

Below is a table outlining operator precedence in C++. Operations listed at the top of the table have the highest precedence while operations listed at the bottom of the table have the lowest precedence. The **associativity** of operations refers to the order with which operations of the same precedence are performed. In this table, operators in the same row have the same precedence.

Precedence	Operator	Symbol	Associativity
1	Scope resolution	::	Left to right
2	Suffix/Postfix Function Call Array Subscripting Element selection by reference Element selection through pointer	++ -- () [] . ->	
3	Prefix Unary Logical NOT and bitwise NOT Type cast Indirection (dereference) Address-of Size-of Dynamic memory allocation Dynamic memory deallocation	++ -- + - ! ~ (type) * & sizeof new, new[] delete, delete[]	Right to left
4	Pointer to member	.* ->*	Left to right

5	Multiplication, division, remainder	* / %	
6	Addition, subtraction	+ -	
7	Bitwise left shift and bitwise right shift	<< >>	
8	Relational Operators	< <= > >=	
9	Equality	== !=	
10	Bitwise AND	&	
11	Bitwise XOR	^	
12	Bitwise OR	|	
13	Logical AND	&&	
14	Logical OR	||	
15	Assignment	=	Right-to-left

Look at the following code:

```
int foo() {
     int expr = 1;
     expr = 2 * expr++;
     return expr;
}
```

If a person looking at this code wasn't aware of operator precedence, he or she might believe that the multiplication of 2 and expr happens first. Thus, the function foo, in their eyes would return 3. However, the postfix incrementation happens first, before the multiplication. Thus, the function returns 4, not 3.

If you wish to override operator precedence, ***placing parentheses around portions of the equation places precedence on those operations***. Parenthesized expressions will be calculated first and then order of operations takes effect.

Exercises

1. Remember in high school when you learned about the formula that allowed you to calculate the roots of a quadratic equation, regardless of what the equation was? For those of you who don't recall the formula, it is the one below:

   $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   Since there is a $\pm$ in the equation, there are potentially two roots of a given quadratic equation. Write two functions that return the two roots, given the coefficients a, b, and c.

```
double getRoot1(double a, double b, double c) {




}

double getRoot2(double a, double b, double c) {




}
```

   Would your functions always return the correct values? Why?

2. Later on during the development cycle, you realize that the same engineer merged a piece of code that seems to be causing some problems:

```
#define FAIL 0
#define SUCCESS 1

int validateUser(string username, string password) {
   int isUser;

   // Call function to authenticate username and password.
   // Assign isUser to be result of authenticate.
   // If authentication fails, print error message

   if (isUser = authenticate(username, password) == FAIL) {
      cout << "Invalid user" << endl;
   }

   return isUser;
}
```

The style that is shown here, assigning a variable within an `if` condition, used to be very common style in C++ programming! You find that in certain situations, the authenticate() function returns FAIL, yet the `validateUser()` function returns `SUCCESS`. Why is this? What can you do to fix it?

3. The leap year is a phenomenon that occurs when fractions of day build up after years and thus adds a full day to a year. However, the formula for calculating a leap year isn't as simple as finding years that are divisible by four. 2000 is a leap year but 1900 isn't. A particular year is a leap year if it is divisible by 4. However, a year isn't a leap year if it is divisible by 100 unless it is also divisible by 400.

    a. Write some pseudo code that will return true if the given integer year is a leap year and false if it is not.

    b. Condense your pseudo code into actual code that uses a ***single*** if-else statement. (Make sure you take into account operator precedence!)

Security Implications

1. On July 22, 1962, Mariner 1 was launched with a Venus flyby mission. Sometime after launch, the rocket made a hard left, which consequently pointed the rocket downwards. 294.5 seconds after launch, a destructive abort was ordered and the rocket exploded.

   Later it was realized that Mariner 1 actually contained 2 errors that resulted in the forced destruction of the rocket. The first error was that the rocket lost contact with the ground guidance signal that supplied the steering commands. This possibility was foreseen. Thus, in the event that radio guidance was lost, the internal guidance computer was supposed to reject spurious signals from the faulty antenna and proceed to its stored directional program. This is the point at which the second error occurred. A mistake in the guidance program loaded aboard the computer allowed flawed signals to command the rocket to veer left and nose down. Although there is no official report that describes the bug, the flawed signals were said to have been caused by an operator precedence error in the guidance program code. This mistake was present in previous successful flights but the portion including the flawed equation had not been needed since there were no radio guidance failures previously. This mistake in the equation cost NASA $18.5 million dollars in 1962, which is equivalent to $135 million dollars today.

   For more information about the Mariner 1 mission see:
   http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=MARIN1
   http://catless.ncl.ac.uk/Risks/5.73.html#subj2
   http://catless.ncl.ac.uk/Risks/9.54.html#subj1.1

2. High frequency trading algorithms take into account the cost of **latency**, or time delay experienced by a system. This is especially important in high frequency trading, where hundreds of thousands of trades occur every second and the slightest difference in price can negatively affect both the trades and the trading company. If a company miscalculates the latency cost of their system, they may be losing out on hundreds of thousands of dollars.

   On August 1, 2012, Knight Group lost approximately $440 million in the first 30 minutes of its business day, lowering the value of KCG stocks by over 70%. Their trading activities at the beginning of this day caused a major disruption in the prices of 148 companies listed in the New York Stock Exchange, thus erroneously affecting the prices of various stocks. For example, because of Knight's trading, shares of Wizzard Software Corporation rose from $3.50 per share to $14.76 per share. While we aren't complete aware of the exact reason why Knight Group made these erroneous trades, we can assume that a mistake in their high frequency trading algorithm caused the issue. Such a mistake may have been an operator precedence error in their latency cost equation.

   Let's say that Knight group used the ***iteration method*** with the following equation to calculate the latency cost:

$$l_{i+1} = l_i - \frac{(a\sqrt{t})l_i \log(\frac{bl_i}{t})}{a\sqrt{t}\,(\log(\frac{bl_i}{t}) + 1)}$$

   where t is the latency, a is the price volatility, and b is what is called the bid-offer spread*, l_i is the i^{th} iteration of the latency cost and l_{i+1} is the $(i+1)^{th}$ iteration of the latency cost.

   This function uses the ***iteration method***. First, we start with i=0: we are given the values for *a*, *b*, and *t* and we simply guess a value for l_0 (to plug into l_i in the right-hand side). Then, we substitute the values into the right-hand side to yield a value for l_1 . Here's where the iteration comes in. We then take the difference $l_1 - l_0$, and if the difference is smaller than a given value ε(epsilon), the value of l_1 is the answer. If the difference is larger than ε, we repeat the equation except we plug l_1 into the right hand side, yielding l_2 , and take the difference $l_2 - l_1$ and compare it to ε.

   Given functions `log(int n)` and `sqrt(float f)` and values for *a*, *b*, *t*, l_0, and ε, write a function that returns the latency cost using the equation above and the iteration method.
   (HINT: you will probably want to use a `while` *loop)*

*Note that the **bid-offer spread** is the difference between the prices quoted for an immediate sale (offer) and an immediate purchase (bid).

```
double latencyCost(double a, double b, double t, double L0,
                   double epsilon) {









}
```

Plug in the following values for a, b, t, L_0, and epsilon into your function. If you've written it correctly, it should be close to the answers that follow. Otherwise, it can converge to the wrong values!

1) a : 5
   b : 4
   t : 10
   L_0 : 1.5
   epsilon: 0.01

   this should converge to about 2.5

2) a : 1
   b : 2
   t : 20
   L_0 : 5
   epsilon: 0.01

   this should converge to about 10

3) a : 3.7
   b : 6.66
   t : 40.4
   L_0 : 4
   epsilon: 0.005

   this should converge to about 6.066

Consider the following function that returns a single iteration of the latency cost:

```
double latencyIteration(double a, double b, double t, double L0) {
      double pi = 3.1415;
      return
        L0 - (a*sqrt(t)*L0*log(L0*b/t) / a*sqrt(t)*(log(L0*b/t)+1));
}
```

It has been deduced that this function is the reason for the $440 million loss for the Knight Group.

1. Why is this function incorrect? What simple revisions could be made to the function that would make it behave correctly?

2. Is the value returned by the function greater or smaller than the actual value returned by the mathematical equation? What would happen if this function were to be used in the iteration method for finding the latency cost?