A Quick Guide to Precedence

Professor Don Colton

Brigham Young University Hawaii

1 What Are We Trying To Do?

When we say something like "three plus two times five" there are two ways to interpret it. We could do the plus first, then the times, for an answer of 25. Or we could do the times first, then the plus, for an answer of 13.

When there is more than one reasonable answer, it is called **ambiguity**.

Humans deal with ambiguity fairly well in most cases. Often we do not even notice it. Common sense is called into play and the alternatives are rated for likelihood or reasonableness. The unlikely alternatives are dropped. If there is only one left, we are happy. If there are two or more reasonable alternatives, we recognize that ambiguity exists.

In a case like $3+2\times5$, there is not much common sense to call upon. So we have commonly accepted rules. These rules are called the rules of **precedence**.

2 Introduction

Precedence comes from the word **precede**, meaning to go before, or to happen first.

For convenience and simplicity, mathematicians have standardized on the precedence of multiplication before addition. It does not have to be that way, but by convention we have agreed as a group to follow that rule.

In the $3+2\times5$ example, this means we multiply 2 and 5 for a result of 10. We then add 3 for a result of 13.

If we want the other meaning, we use parentheses to alter the precedence. We say $(3+2)\times5$, meaning add 3 and 2 for a result of 5. Then multiply that by 5 for a result of 25.

Writing without parentheses is shorter. Shorter is more desirable because it is less work. It is less work to write, but it does require us to memorize the rules of precedence so we will get the same answer as everyone else.

3 The Multiply Class

By convention (by common agreement), multiply, divide, and remainder all have the same precedence, and we do the operations within that class from left to right. (All of the examples on the quizzes have a left-to-right precedence within the same precedence class. There are some classes where the precedence is right to left, but those will not be on our quiz.)

`8/4/2` means 8 divided by 4, then that divided by 2. `8/4=2` and `2/2=1` so the answer is 1.

If we did not follow the left-to-right precedence, we might have figured it like this: `4/2=2` and `8/2=4` for an answer of 4.

We can see that the order is significant. It makes a difference.

3.1 The Multiply Symbol

In computer programming, we often use the asterisk (`*`) to represent multiplication. In arithmetic we are accustomed to using an x ($\times$) to represent multiplication, but in computer programming we want the letter `x` to be a letter, so the asterisk has been assigned the meaning of multiply.

We say `3*5` is three times five, or 15.

3.2 Integer Divide

For the quizzes in this course, when we divide we are using integer division. Normally when we think of division, we think of calculator division, where `5/2` is `2.5`. But for this course, `5/2=2r1`, five divided by two is two with a remainder of one.

`11/4` means eleven divided by four, but just the whole-number **quotient** is kept, which is 2.

3.3 Remainder

The percent sign (`%`) is used to indicate division, but we only want the remainder. `11%4` means eleven divided by four, but just the remainder is kept, which is 3.

Remainder is sometimes called **modulus** or **mod**. Division with remainders is sometimes called "clock arithmetic" because 11 o'clock plus three hours equals 2 o'clock on a 12-hour clock: `(11+3)%12` is 2.

3.4 Cookies and Children

One example that I frequently use to describe integer division is my "cookies and children" example. In this example, we imagine that we have, say, eleven cookies and four children. We wish to divide the cookies.

Now, because we are dealing with children, there are several rules we must follow. (1) We must give each child the maximum number of cookies. (2) We must give each child the same number of cookies as every other child. (3) We must not break any of the cookies. (4) Any extra cookies are kept by the "mommy."

`11/4` means eleven cookies, four children. Each child gets two cookies and there are three cookies left over for the mommy. The answer is 2 (the number each child gets).

`11%4` means eleven cookies, four children. Each child gets two cookies and there are three cookies left over for the mommy. The answer is 3 (the number the mommy gets).

4 Precedence Tables

Here is the precedence table for the operators used on the first quiz.

rank	operators	dir
1	`*` `/` `%`	l-r
2	`+` `-`	l-r

This table means that the top-ranking precedence, the number one precedence, is shared by multiply (`*`), divide (`/`), and remainder (`%`). Within that class, precedence is left-to-right (l-r).

Similarly, the second-ranking (number two) precedence is shared by add (`+`) and subtract (`-`). Within that class, precedence is left-to-right (l-r).

Here is the full precedence table for the operators used on the quizzes after the first.

rank	operators	dir
1	`*` `/` `%`	l-r
2	`+` `-`	l-r
3	`<` `<=` `>` `>=`	l-r
4	`==` `!=`	l-r
5	`&&`	l-r
6	`||`	l-r

5 The Other Operators

This table introduces several other operators. The meaning of each operator is explained in a section below.

5.1 Less Than, Greater Than

At rank three, we have less than (`<`), less than or equal to (`<=`), greater than (`>`), and greater than or equal to (`>=`).

If we say `a<b`, we mean "is `a` less than `b`?" The answer is yes (true) or no (false). When the answer is yes, we use `1` (one), representing true. When the answer is no, we use `0` (zero), representing false.

Thus, `3<5` is `1`, because three is less than five, true.

We use `<=` for "less than or equal to" because the $\leq$ symbol is not found on most computer keyboards.

We use `>=` for "greater than or equal to" because the $\geq$ symbol is not found on most computer keyboards.

5.2 Equal, Not Equal

At rank four, we have equal (`==`) and not equal (`!=`). When the answer is yes, we use `1` (one), representing true. When the answer is no, we use `0` (zero), representing false.

We use `==` for "equal to" because the `=` symbol is already being used for another purpose. `a=b` means to copy the value from `b` and store it into variable `a`. `a==b` means to compare `a` and `b` and answer with `1` if they are the same (equal), and with `0` if they are different (not equal).

We use `!=` for "not equal to" because the $\neq$ symbol is not found on most computer keyboards. Some computer languages use `<>` (less than or greater than) to mean not equal, but `!=` will be used in this course.

5.3 Logical And

At rank five, we have "and" (`&&`), also called "logical and" (as distinct from "bitwise and"). When the an-

swer is yes, we use `1` (one), representing true. When the answer is no, we use `0` (zero), representing false.

What does "and" mean? We are combining two quantities. How do we do it?

When we say "the sky is blue and I am happy," we are combining two statements. The first is "the sky is blue." The second is "I am happy." We can determine the truth of each statement separately. Look outside. Is the sky blue? If so, then the truth value of the first statement is `1` (meaning "true").

When both statements are true, the resulting compound statement is true. If either statement is false, the resulting compound statement is false. That is the way that "and" works.

The truth table for "and" looks like this:

expression	value
`1 && 1`	`1`
`1 && 0`	`0`
`0 && 1`	`0`
`0 && 0`	`0`

We also have a convention for what to do if the input numbers are not one or zero. All non-zero numbers are treated as "true." Only zero is treated as "false." In particular, both positive **and** negative numbers are "true."

Thus, `5&&0` is false, `-7&&12` is true, and `0&&4` is false.

5.4 Logical Or

At rank six, we have "or" (`||`), also called "logical or" (as distinct from "bitwise or"). When the answer is yes, we use `1` (one), representing true. When the answer is no, we use `0` (zero), representing false.

What does "or" mean? We are combining two quantities. How do we do it?

When we say "the sky is blue or I am happy," we are combining two statements. When either statement is true, the resulting compound statement is true. If both statements are false, the resulting compound statement is false. That is the way that "or" works.

The truth table for "or" looks like this:

expression	value
`1 || 1`	`1`
`1 || 0`	`1`
`0 || 1`	`1`
`0 || 0`	`0`

Once again, non-zero inputs are treated as "true." Only zero is treated as "false."

Thus, `5||0` is true, `-7||12` is true, `0||4` is true, and `0||0` is false.

This version of "or" is also called **inclusive or** because it includes the case where both statements are true. In legal writing it is often phrased "and/or." There is another version of "or" called **exclusive or** whose truth table looks like this:

expression	value
`1 ^ 1`	`0`
`1 ^ 0`	`1`
`0 ^ 1`	`1`
`0 ^ 0`	`0`

With "exclusive or" (xor) the result is true when one statement or the other, but not both, is true. Exclusive or is also called **parity**. "and" and "or" are available as both logical operators and as bitwise operators, but "xor" is often restricted to the bitwise domain.

5.5 Bitwise Operators

We have referred to "bitwise and," "bitwise or," and "bitwise xor" in the paragraphs above. In this section we will touch very briefly on the difference between bitwise and logical.

With "logical" we regard the number as a whole. The number is either true (non-zero) or false (zero). With "bitwise" the number is regarded as a string of bits. Two numbers become two strings of bits, and the operator (and, or, xor) is applied to pairs of bits in each number in order.

It is more clear in binary. If we say "`5||3`" we mean "true or true." If we say "`5|3`" we mean "`101|011`" for which the answer is "`111`" (7). That is, we combine the first bits (1 or 0), the second bits (0 or 1), and the third bits (1 or 1) to get the answer.

5.6 Short Circuits

In the case of "logical and" and "logical or" there is a shortcut we can frequently take to reach an answer. This shortcut is called a "short circuit."

To better explain this, consider the following mathematical calculation:

$$13534582934581254 \times 0$$

Can you determine the answer? Can you do so without the aid of a calculator?

We know that when any number is multiplied by zero, the answer is zero. This is a shortcut we can use to determine the result. We may even be able to determine the answer to a calculation before all the parts are known. For example, if I say: $0 \times 1353\ldots$ you can tell me the result before I provide the rest of that big and ugly number. You would have "short circuited" the calculation.

When we use "and" to combine two numbers, if the first number is zero, we know the answer will be false (zero).

When we use "or" to combine two numbers, if the first number is non-zero, we know the answer will be true (one).

In both of these cases, we do not even bother to calculate the rest of the formula. By the rules of computing, we are actually forbidden to calculate the rest of the formula. We must stop when we know the answer.

When does this matter? Here is an example.

"`5/0`" is `error` because dividing by zero is illegal.

"`5%0`" is `error` because dividing by zero is illegal.

"`5%0 || 7`" is `error` because dividing by zero is illegal.

"`7 || 5%0`" is `1`. The divide by zero never happens because short-circuiting tells us the answer: `true or` anything is `true`.

"`5/0 && 0`" is `error` because dividing by zero is illegal.

"`0 && 5/0`" is `0`. The divide by zero never happens because short-circuiting tells us the answer: `false and` anything is `false`.

There is no short-circuit for "exclusive or."

5.7 Beware Language Differences

C and Perl behave slightly differently. In C, `5||3` is true, represented by 1. In Perl the answer is also true, but it is represented by 5, the number at which short circuit took over. Language designers tend to agree on basic principles but vary on some details. When programming in a new language it is smart to verify that things work as expected before writing much code.