**Infix, Prefix and Postfix Expressions**

When you write an arithmetic expression such as B \* C, the form of the expression provides you with information so that you can interpret it correctly. In this case we know that the variable B is being multiplied by the variable C since the multiplication operator \* appears between them in the expression. This type of notation is referred to as **infix** since the operator is *in between* the two operands that it is working on.

Consider another infix example, A + B \* C. The operators + and \* still appear between the operands, but there is a problem. Which operands do they work on? Does the + work on A and B or does the \* take B and C? The expression seems ambiguous.

In fact, you have been reading and writing these types of expressions for a long time and they do not cause you any problem. The reason for this is that you know something about the operators + and \*. Each operator has a **precedence** level. Operators of higher precedence are used before operators of lower precedence. The only thing that can change that order is the presence of parentheses. The precedence order for arithmetic operators places multiplication and division above addition and subtraction. If two operators of equal precedence appear, then a left-to-right ordering or associativity is used.

Let’s interpret the troublesome expression A + B \* C using operator precedence. B and C are multiplied first, and A is then added to that result. (A + B) \* C would force the addition of A and B to be done first before the multiplication. In expression A + B + C, by precedence (via associativity), the leftmost + would be done first.

Although all this may be obvious to you, remember that computers need to know exactly what operators to perform and in what order. One way to write an expression that guarantees there will be no confusion with respect to the order of operations is to create what is called a **fully parenthesized** expression. This type of expression uses one pair of parentheses for each operator. The parentheses dictate the order of operations; there is no ambiguity. There is also no need to remember any precedence rules.

The expression A + B \* C + D can be rewritten as ((A + (B \* C)) + D) to show that the multiplication happens first, followed by the leftmost addition. A + B + C + D can be written as (((A + B) + C) + D) since the addition operations associate from left to right.

There are two other very important expression formats that may not seem obvious to you at first. Consider the infix expression A + B. What would happen if we moved the operator before the two operands? The resulting expression would be + A B. Likewise, we could move the operator to the end. We would get A B +. These look a bit strange.

These changes to the position of the operator with respect to the operands create two new expression formats, **prefix** and **postfix**. Prefix expression notation requires that all operators precede the two operands that they work on. Postfix, on the other hand, requires that its operators come after the corresponding operands. A few more examples should help to make this a bit clearer (see [Table 2](https://runestone.academy/ns/books/published/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html#tbl-example1)).

A + B \* C would be written as + A \* B C in prefix. The multiplication operator comes immediately before the operands B and C, denoting that \* has precedence over +. The addition operator then appears before the A and the result of the multiplication.

In postfix, the expression would be A B C \* +. Again, the order of operations is preserved since the \* appears immediately after the B and the C, denoting that \* has precedence, with + coming after. Although the operators moved and now appear either before or after their respective operands, the order of the operands stayed exactly the same relative to one another.

| **Table 2: Examples of Infix, Prefix, and Postfix** | | |
| --- | --- | --- |
| **Infix Expression** | **Prefix Expression** | **Postfix Expression** |
| A + B | + A B | A B + |
| A + B \* C | + A \* B C | A B C \* + |

Now consider the infix expression (A + B) \* C. Recall that in this case, infix requires the parentheses to force the performance of the addition before the multiplication. However, when A + B was written in prefix, the addition operator was simply moved before the operands, + A B. The result of this operation becomes the first operand for the multiplication. The multiplication operator is moved in front of the entire expression, giving us \* + A B C. Likewise, in postfix A B + forces the addition to happen first. The multiplication can be done to that result and the remaining operand C. The proper postfix expression is then A B + C \*.

Consider these three expressions again (see Table 3). Something very important has happened. Where did the parentheses go? Why don’t we need them in prefix and postfix? The answer is that the operators are no longer ambiguous with respect to the operands that they work on. Only infix notation requires the additional symbols. The order of operations within prefix and postfix expressions is completely determined by the position of the operator and nothing else. In many ways, this makes infix the least desirable notation to use.

| **Table 3: An Expression with Parentheses** | | |
| --- | --- | --- |
| **Infix Expression** | **Prefix Expression** | **Postfix Expression** |
| (A + B) \* C | \* + A B C | A B + C \* |

Table 4 shows some additional examples of infix expressions and the equivalent prefix and postfix expressions. Be sure that you understand how they are equivalent in terms of the order of the operations being performed.

| **Table 4: Additional Examples of Infix, Prefix, and Postfix** | | |
| --- | --- | --- |
| **Infix Expression** | **Prefix Expression** | **Postfix Expression** |
| A + B \* C + D | + + A \* B C D | A B C \* + D + |
| (A + B) \* (C + D) | \* + A B + C D | A B + C D + \* |
| A \* B + C \* D | + \* A B \* C D | A B \* C D \* + |
| A + B + C + D | + + + A B C D | A B + C + D + |

**Conversion of Infix Expressions to Prefix and Postfix**

So far, we have used ad hoc methods to convert between infix expressions and the equivalent prefix and postfix expression notations. As you might expect, there are algorithmic ways to perform the conversion that allow any expression of any complexity to be correctly transformed.

The first technique that we will consider uses the notion of a fully parenthesized expression that was discussed earlier. Recall that A + B \* C can be written as (A + (B \* C)) to show explicitly that the multiplication has precedence over the addition. On closer observation, however, you can see that each parenthesis pair also denotes the beginning and the end of an operand pair with the corresponding operator in the middle.

Look at the right parenthesis in the subexpression (B \* C) above. If we were to move the multiplication symbol to that position and remove the matching left parenthesis, giving us B C \*, we would in effect have converted the subexpression to postfix notation. If the addition operator were also moved to its corresponding right parenthesis position and the matching left parenthesis were removed, the complete postfix expression would result (see Figure 6).

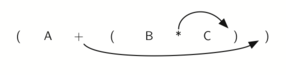


Figure 6: Moving Operators to the Right for Postfix Notation

If we do the same thing but instead of moving the symbol to the position of the right parenthesis, we move it to the left, we get prefix notation (see Figure 7). The position of the parenthesis pair is actually a clue to the final position of the enclosed operator.

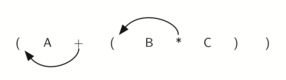


Figure 7: Moving Operators to the Left for Prefix Notation

So in order to convert an expression, no matter how complex, to either prefix or postfix notation, fully parenthesize the expression using the order of operations. Then move the enclosed operator to the position of either the left or the right parenthesis depending on whether you want prefix or postfix notation.

Here is a more complex expression: (A + B) \* C - (D - E) \* (F + G). Figure 8 shows the conversion to postfix and prefix notations.

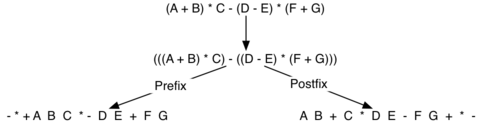


Figure 8: Converting a Complex Expression to Prefix and Postfix Notations

## General Infix-to-Postfix Conversion

We need to develop an algorithm to convert any infix expression to a postfix expression. To do this we will look closer at the conversion process.

Consider once again the expression A + B \* C. As shown above, A B C \* + is the postfix equivalent. We have already noted that the operands A, B, and C stay in their relative positions. It is only the operators that change position. Let’s look again at the operators in the infix expression. The first operator that appears from left to right is +. However, in the postfix expression, + is at the end since the next operator, \*, has precedence over addition. The order of the operators in the original expression is reversed in the resulting postfix expression.

See lecture slides for algorithm. Keep in mind that there are various algoritms similar to the one we have on our web site.