The prefix and postfix notations are not really as awkward to use as they might look.

For example, a C function to return the sum of two variables A and B (passed as argument)

is called or invoked by the instruction:

add(A, B)

Note that the operator *add* (name of the function) precedes the operands A and B.

Because the postfix notation is most suitable for a computer to calculate any expression

(due to its reverse characteristic), and is the universally accepted notation for designing

Arithmetic and Logical Unit (ALU) of the CPU (processor).

Therefore it is necessary to

study the postfix notation. Moreover the postfix notation is the way computer looks towards

arithmetic expression, any expression entered into the computer is first converted

into postfix notation, stored in stack and then calculated. In the preceding sections we will

study the conversion of the expression from one notation to other.

**Advantages of using postfix notation**

Human beings are quite used to work with mathematical expressions in *infix* notation,

which is rather complex. One has to remember a set of nontrivial rules while using

this notation and it must be applied to expressions in order to determine the final value.

These rules include precedence, BODMAS, and associativity.

Using infix notation, one cannot tell the order in which operators should be applied.

Whenever an infix expression consists of more than one operator, the precedence rules

(BODMAS) should be applied to decide which operator (and operand associated with that

operator) is evaluated first.

But in a postfix expression operands appear before the operator,

so there is no need for operator precedence and other rules. As soon as an operator

appears in the postfix expression during scanning of postfix expression the topmost operands

are popped off and are calculated by applying the encountered operator. Place the

result back onto the stack; likewise at the end of the whole operation the final result will

be there in the stack.

**Notation Conversions**

Let A + B \* C be the given expression, which is an infix notation. To calculate this

expression for values 4, 3, 7 for A, B, C respectively we must follow certain rule (called

BODMAS in general mathematics) in order to have the right result. For example:

A + B \* C = 4 + 3 \* 7 = 7 \* 7 = 49

The answer is **not correct**; multiplication is to be done before the addition, because

multiplication has higher precedence over addition. This means that an expression is

calculated according to the operator’s precedence not the order as they look like. The error

in the above calculation occurred, since there were no braces to define the precedence of

the operators. Thus expression A + B \* C can be interpreted as A + (B \* C).

Using this

alternative method we can convey to the computer that multiplication has higher precedence

over addition.

*Operator precedence*

Exponential operator ^ Highest precedence

Multiplication/Division \*, / Next precedence

Addition/Subtraction +, - Least precedence