Interpreter

The interpreter performs the following general operations:

- Carry out lexical analysis (scanner)
- Carry out syntax analysis, this includes syntax checking (parser),
- Directly or indirectly convert the input program into an internal form (parser).
- Execute the internal form program statement by statement (the executer).

The parser's main goal is to recognize valid sentences (as specified in the language grammar) in the input program.

At the implementation level, the parser is called once to analyze syntactically the entire source program in the input file. The parser calls the scanner every time in needs a token from the input file.

A table or stack is used to store the values of all variables defined in the source program. Depending on the variable types used this might be organized in a simple manner as separate tables, one for each type.

For the internal form, RPN or reverse polish notation (postfix) is often used. This internal form of the program is stored in an integer array, P. When executing the internal form, an index of this array is used as an "instruction counter", and a stack is used.

The following scheme may be useful for simple interpreters. The array P with the internal form stores the code of an operand or operator. If it is an identifier or constant, the next element stores the current value of the identifier or constant.

The interpreter has a case statement (in a loop) with one case for each of the possible different operators and types of operands.

Symbol Table

I recommend using two separate tables:

- The keyword table, handled by the scanner
- The identifier table, handled by the parser. This table includes several attributes for every identifier, this includes the current value of the identifier. Constants can also be stored in this table.

Two functions are used to manipulate these tables:

int insert (string name, int token), used with the identifier table (only)

int lookup (string name), used with both tables

The two functions return the index value of the entry in the table.

Reverse Polish Notation (optional)

In reverse Polish notation the operators follow their operands; for instance, to add 3 and 4, one would write “3 4 +” rather than “3 + 4”. If there are multiple operations, the operator is given immediately after its second operand; so the expression written “3 – 4 + 5” in conventional notation would be written “3 4 – 5 +” in RPN: 4 is first subtracted from 3, then 5 added to it.

An advantage of RPN is that it obviates the need for parentheses that are required by infix. While “3 – 4 * 5” can also be written “3 – (4 * 5)”, that means something quite different from “(3 – 4) * 5”. In postfix, the former could be written “3 4 5 * -“, which unambiguously means “3 (4 5 *) -” which reduces to “3 20 -“; the latter could be written “3 4 – 5 *” (or 5 3 4 – *, if keeping similar formatting), which unambiguously means “(3 4 -) 5 *”.