The computer usually evaluates an arithmetic expression written in infix notation in two steps. First, it converts the expression to postfix notation, and then it evaluates the postfix expression. In each step, the stack is the main tool that is used to accomplish the given task. We illustrate these applications of stacks in reverse order. That is, first we show how stacks are used to evaluate postfix expressions, and then we show how stacks are used to transform infix expressions into postfix expressions.

Evaluation of a Postfix Expression

Suppose P is an arithmetic expression written in postfix notation. The following algorithm, which uses a STACK to hold operands, evaluates P.

Algorithm 6.3: This algorithm finds the VALUE of an arithmetic expression P written in postfix notation.

1. Add a right parenthesis ")" at the end of P. [This acts as a sentinel.]
2. Scan P from left to right and repeat Steps 3 and 4 for each element of P until the sentinel ")" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator $\otimes$ is encountered, then:
   (*a*) Remove the two top elements of STACK, where A is the top element and B is the next-to-top element.
   (*b*) Evaluate $B \otimes A$.
   (*c*) Place the result of (*b*) back on STACK.
   [End of If structure.]
   [End of Step 2 loop.]
5. Set VALUE equal to the top element on STACK.
6. Exit.

We note that, when Step 5 is executed, there should be only one number on STACK.

EXAMPLE 6.5

Consider the following arithmetic expression P written in postfix notation:

P: 5, 6, 2, +, *, 12, 4, /, −

(Commas are used to separate the elements of P so that 5, 6, 2 is not interpreted as the number 562.) The

Symbol Scanned		STACK
(1)	5	5
(2)	6	5, 6
(3)	2	5, 6, 2
(4)	+	5, 8
(5)	*	40
(6)	12	40, 12
(7)	4	40, 12, 4
(8)	/	40, 3
(9)	−	37
(10)	)	

Fig. 6-7

equivalent infix expression Q follows:

Q: 5 * (6 + 2) − 12 / 4

Note that parentheses are necessary for the infix expression Q but not for the postfix expression P.

We evaluate P by simulating Algorithm 6.3. First we add a sentinel right parenthesis at the end of P to obtain

P:	5,	6,	2,	+,	*,	12,	4,	/,	−,	)
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)

The elements of P have been labeled from left to right for easy reference. Figure 6-7 shows the contents of STACK as each element of P is scanned. The final number in STACK, 37, which is assigned to VALUE when the sentinel ")" is scanned, is the value of P.

Transforming Infix Expressions into Postfix Expressions

Let Q be an arithmetic expression written in infix notation. Besides operands and operators, Q may also contain left and right parentheses. We assume that the operators in Q consist only of exponentiations (↑), multiplications (*), divisions (/), additions (+) and subtractions (−), and that they have the usual three levels of precedence as given above. We also assume that operators on the same level, including exponentiations, are performed from left to right unless otherwise indicated by parentheses. (This is not standard, since expressions may contain unary operators and some languages perform the exponentiations from right to left. However, these assumptions simplify our algorithm.)

The following algorithm transforms the infix expression Q into its equivalent postfix expression P. The algorithm uses a stack to temporarily hold operators and left parentheses. The postfix expression P will be constructed from left to right using the operands from Q and the operators which are removed from STACK. We begin by pushing a left parenthesis onto STACK and adding a right parenthesis at the end of Q. The algorithm is completed when STACK is empty.

Algorithm 6.4: POLISH(Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto STACK, and add ")" to the end of Q.
2. Scan Q from left to right and repeat Steps 3 to 6 for each element of Q until the STACK is empty:
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered, push it onto STACK.
5. If an operator ⊗ is encountered, then:
   (*a*) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than ⊗.
   (*b*) Add ⊗ to STACK.
   [End of If structure.]
6. If a right parenthesis is encountered, then:
   (*a*) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.
   (*b*) Remove the left parenthesis. [Do not add the left parenthesis to P.]
   [End of If structure.]

   [End of Step 2 loop.]
7. Exit.

The terminology sometimes used for Step 5 is that ⊗ will "sink" to its own level.

EXAMPLE 6.6

Consider the following arithmetic infix expression Q:

Q: A + (B * C − (D / E ↑ F) * G) * H

We simulate Algorithm 6.4 to transform Q into its equivalent postfix expression P.
First we push "(" onto STACK, and then we add ")" to the end of Q to obtain:

Q:	A	+	(	B	*	C	−	(	D	/	E	↑	F	)	*	G	)	*	H	)
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)

The elements of Q have now been labeled from left to right for easy reference. Figure 6-8 shows the status of STACK and of the string P as each element of Q is scanned. Observe that

(1) Each operand is simply added to P and does not change STACK.

(2) The subtraction operator (−) in row 7 sends * from STACK to P before it (−) is pushed onto STACK.

(3) The right parenthesis in row 14 sends ↑ and then / from STACK to P, and then removes the left parenthesis from the top of STACK.

(4) The right parenthesis in row 20 sends * and then + from STACK to P, and then removes the left parenthesis from the top of STACK.

After Step 20 is executed, the STACK is empty and

P: A B C * D E F ↑ / G * − H * +

which is the required postfix equivalent of Q.

Symbol Scanned	STACK	Expression P
(1) A	(	A
(2) +	(+	A
(3) (	(+ (	A
(4) B	(+ (	A B
(5) *	(+ (*	A B
(6) C	(+ (*	A B C
(7) −	(+ (−	A B C *
(8) (	(+ (− (	A B C *
(9) D	(+ (− (	A B C * D
(10) /	(+ (− (/	A B C * D
(11) E	(+ (− (/	A B C * D E
(12) ↑	(+ (− (/ ↑	A B C * D E
(13) F	(+ (− (/ ↑	A B C * D E F
(14))	(+ (−	A B C * D E F ↑ /
(15) *	(+ (− *	A B C * D E F ↑ /
(16) G	(+ (− *	A B C * D E F ↑ / G
(17))	(+	A B C * D E F ↑ / G * −
(18) *	(+ *	A B C * D E F ↑ / G * −
(19) H	(+ *	A B C * D E F ↑ / G * − H
(20))		A B C * D E F ↑ / G * − H * +

Fig. 6-8

(*b*) Using the infix expression, we obtain:

$$P = 12/(7-3) + 2*(1+5) = 12/4 + 2*6 = 3 + 12 = 15$$

6.8 Consider the postfix expression P in Prob. 6.7. Evaluate P using Algorithm 6.3.

First add a sentinel right parenthesis at the end of P to obtain:

P: 12, 7, 3, −, /, 2, 1, 5, +, *, +,)

Scan P from left to right. If a constant is encountered, put it on a stack, but if an operator is encountered, evaluate the two top constants on the stack. Figure 6-24 shows the contents of STACK as each element of P is scanned. The final number, 15, in STACK, when the sentinel right parenthesis is scanned, is the value of P. This agrees with the result in Prob. 6.7(*b*).

Symbol	STACK
12	12
7	12, 7
3	12, 7, 3
−	12, 4
/	3
2	3, 2
1	3, 2, 1
5	3, 2, 1, 5
+	3, 2, 6
*	3, 12
+	15
)	15

Fig. 6-24

6.9 Consider the following infix expression Q:

$$Q: \quad ((A+B)*D) \uparrow (E-F)$$

Use Algorithm 6.4 to translate Q into its equivalent postfix expression P.

First push a left parenthesis onto STACK, and then add a right parenthesis to the end of Q to obtain

Q: ((A + B) * D) ↑ (E − F))

(Note that Q now contains 16 elements.) Scan Q from left to right. Recall that (1) if a constant is encountered, it is added to P; (2) if a left parenthesis is encountered, it is put on the stack; (3) if an operator is encountered, it "sinks" to its own level; and (4) if a right parenthesis is encountered, it "sinks" to the first left parenthesis. Figure 6-25 shows pictures of STACK and the string P as each element of Q is scanned. When STACK is empty, the final right parenthesis has been scanned and the result is

P: A B + D * E F − ↑

which is the required postfix equivalent of Q.

6.10 Translate, by inspection and hand, each infix expression into its equivalent prefix expression:

(*a*) $(A-B)*(D/E)$

(*b*) $(A+B \uparrow D)/(E-F)+G$