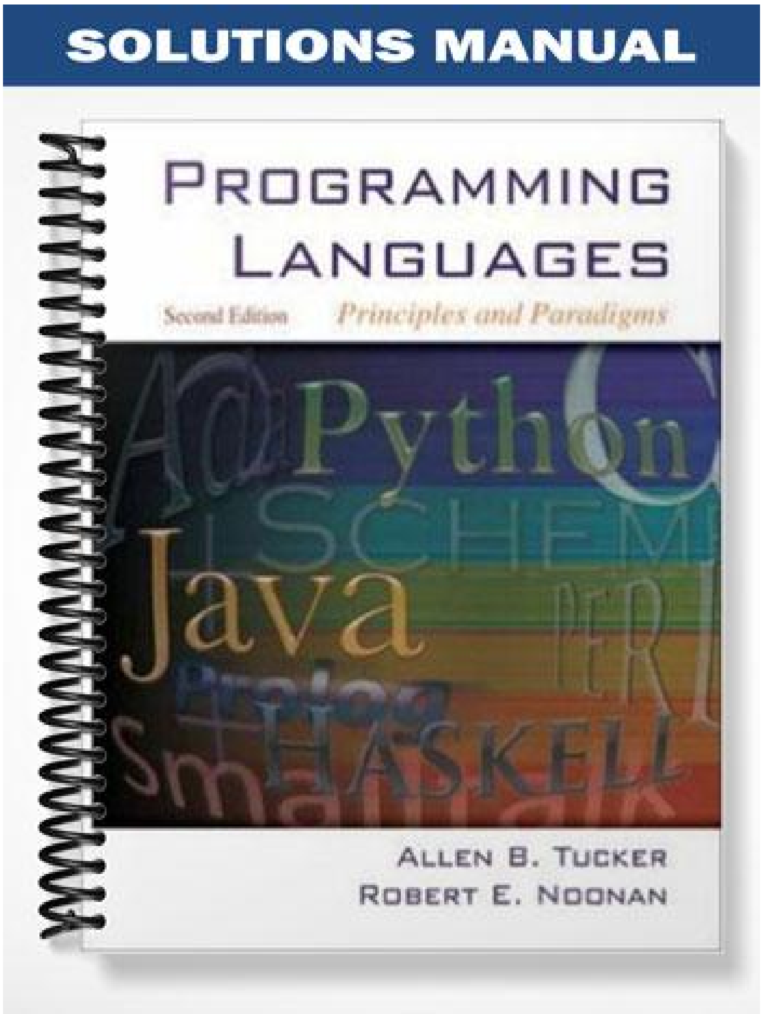


SOLUTIONS MANUAL

PROGRAMMING LANGUAGES

Second Edition *Principles and Paradigms*

ALLEN B. TUCKER
ROBERT E. NOONAN

2

Syntax

Exercises

2.1

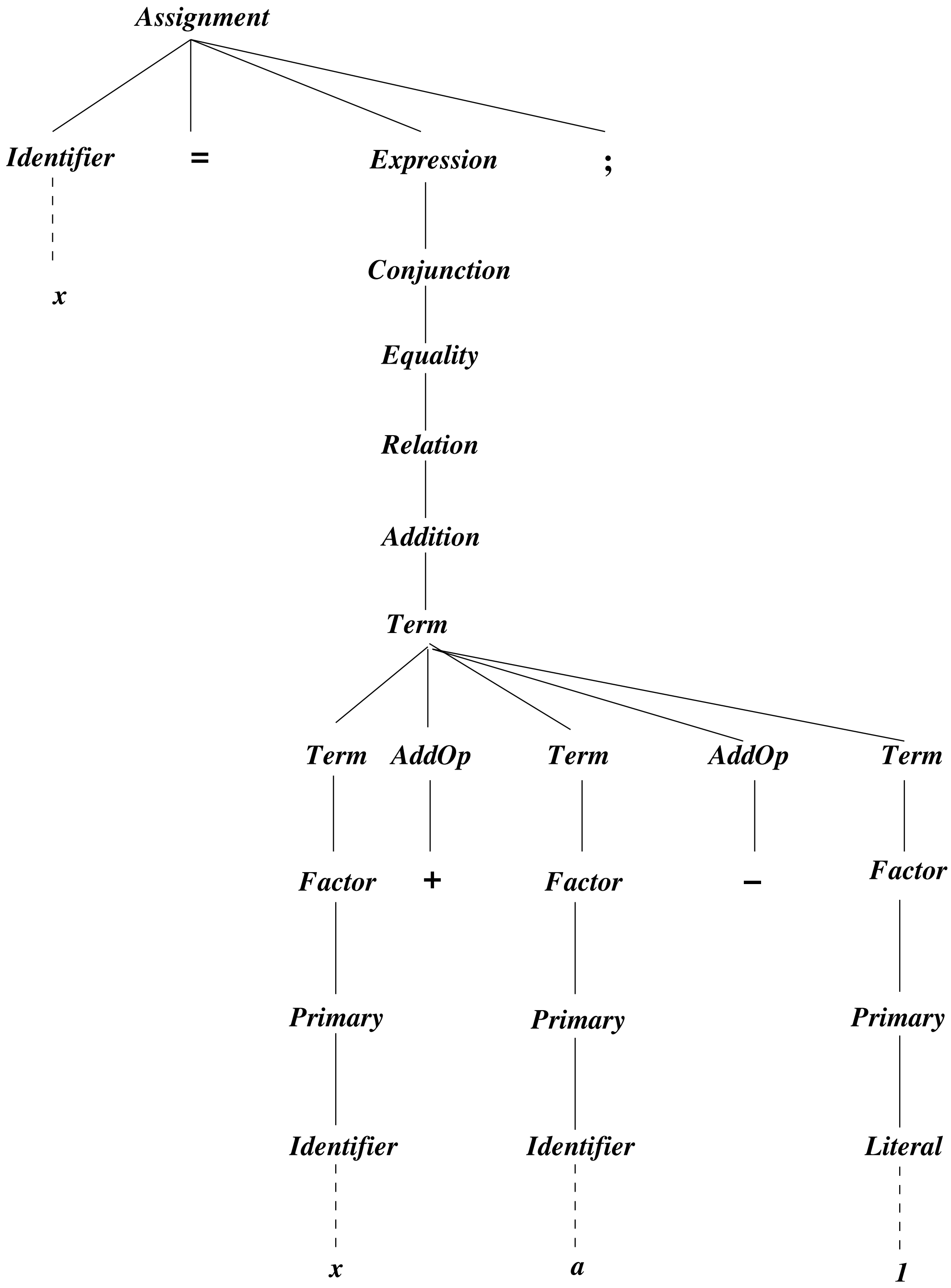
$$\begin{aligned} \mathit{Integer} &\Rightarrow \mathit{Integer\ Digit} \Rightarrow \mathit{Integer\ Digit\ Digit} \Rightarrow \mathit{Integer\ Digit\ Digit\ Digit} \\ &\Rightarrow \mathit{Digit\ Digit\ Digit\ Digit} \Rightarrow 4\ \mathit{Digit\ Digit\ Digit} \Rightarrow 4\ 5\ \mathit{Digit\ Digit} \\ &\Rightarrow 4\ 5\ 2\ \mathit{Digit} \Rightarrow 4\ 5\ 2\ 0 \end{aligned}$$

2.2 This has the same number of steps as above, except that each digit is immediately derived when it appears. E.g., $\mathit{Integer} \Rightarrow \mathit{IntegerDigit} \Rightarrow \mathit{Integer}\ 0 \Rightarrow ...$

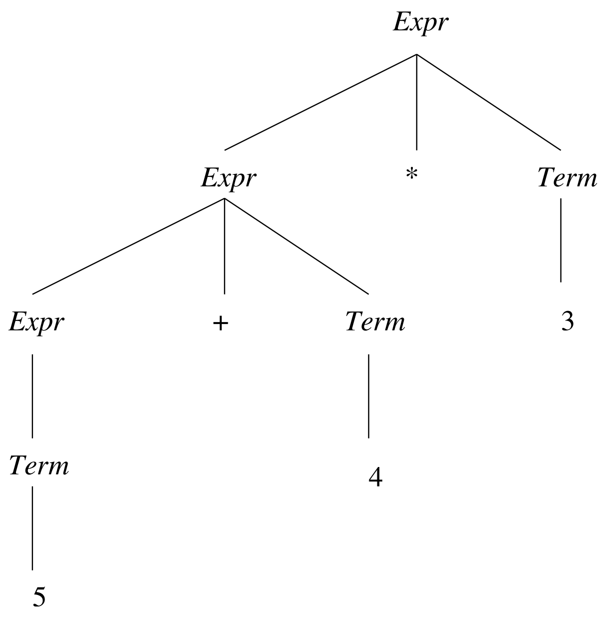
2.3 $\mathit{Identifier} \Rightarrow \mathit{Letter\ Digit\ Letter} \Rightarrow a\ \mathit{Digit\ Letter} \Rightarrow a\ 2\ \mathit{Letter} \Rightarrow a\ 2\ i$

2.4 $\mathit{Identifier} \Rightarrow \mathit{Letter\ Digit\ Letter} \Rightarrow \mathit{Letter\ Digit}\ i \Rightarrow \mathit{Letter}\ 2\ i \Rightarrow a\ 2\ i$

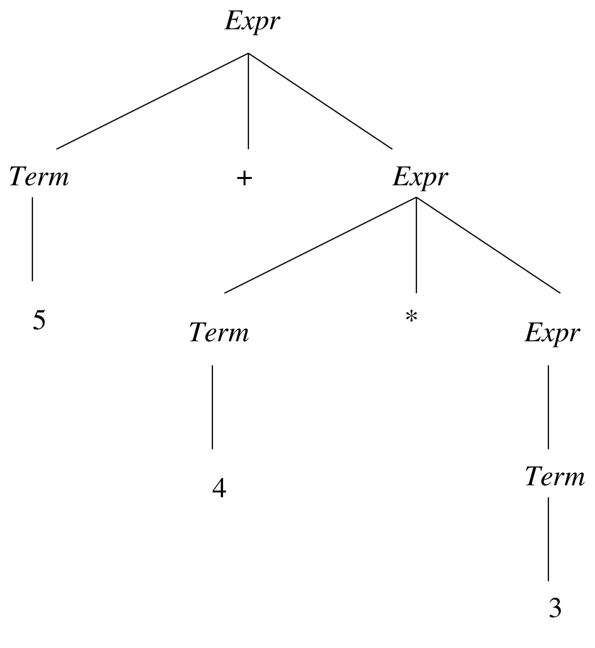
2.5 The solution to part (a) appears below. The solutions to parts (b) and (c) are simple variations of this one.



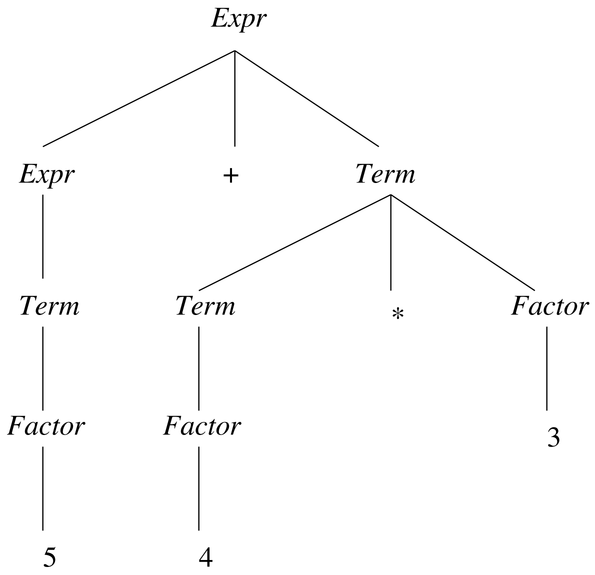
2.6 The solution to part (a) appears below. The solutions to parts (b) and (c) are simple variations of this one.



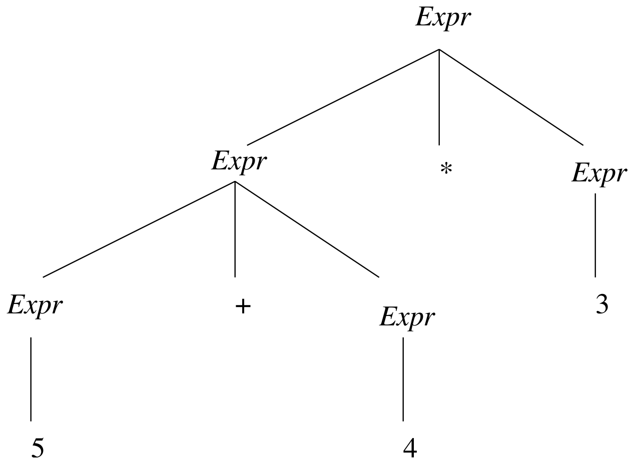
2.7 The solution to part (a) appears below. The solutions to parts (b) and (c) are simple variations of this one.



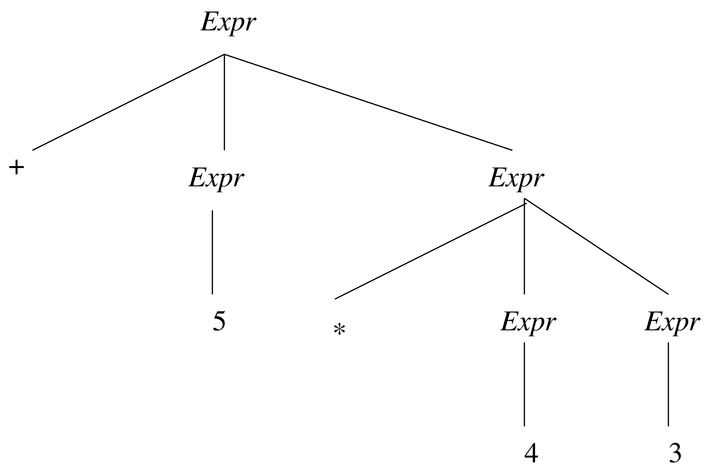
2.8 The solution to part (a) appears below. The solutions to parts (b) and (c) are simple variations of this one.



2.9 There are two solutions to part (a), one of which appears below. The other is a simple variation of this one. (Note that this grammar is ambiguous.) The solutions to parts (b) and (c) are also similar.



2.10 The solution to part (a) appears below. The solutions to parts (b) and (c) are simple variations of this one.



2.11 To derive the *Expression* 5 - 4 + 3, we must use the rules $Expression \rightarrow Expression - Term$ and $Expression \rightarrow Expression + Term$. (These are the only ones that generate the - and + signs, respectively.) So the first three steps in the derivation give:

$Expression \Rightarrow Expression + Term \Rightarrow Expression - Term + Term \Rightarrow Term - Term + Term$

Since no later step in the derivation can reduce the length of this string, each of the three instances of *Term* must derive one of the constants 5, 4, and 3, respectively.

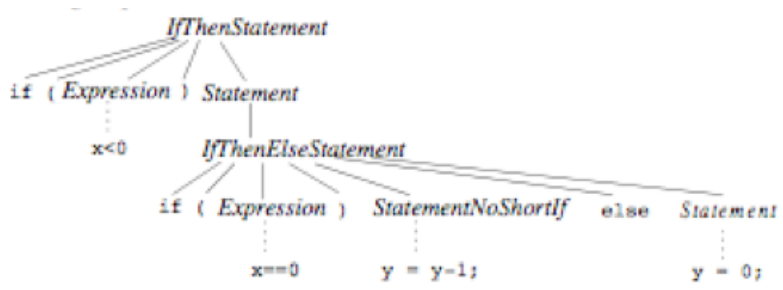
2.12 Consider the Java rules:

$$\begin{aligned} IfThenStatement &\rightarrow \texttt{if}\ (\ Expression\ )\ Statement \\ IfThenElseStatement &\rightarrow \texttt{if}\ (\ Expression\ )\ StatementNoShortIf\ \texttt{else}\ Statement \end{aligned}$$

These rules force the innermost *IfStatement* in

`if (x < 0) if (x == 0) y = y - 1; else y = 0;`

not to have an else part, so that the only feasible parse tree for this statement is has the following shape:



2.13 The rules can be changed by adding the delimiter `fi` as shown below:

$$\begin{aligned} \mathit{IfStatement} &\rightarrow \texttt{if}\ (\ \mathit{Expression}\ )\ \mathit{Statement}\ \texttt{fi}\ | \\ &\rightarrow \texttt{if}\ (\ \mathit{Expression}\ )\ \mathit{Statement}\ \texttt{else}\ \mathit{Statement}\ \texttt{fi} \end{aligned}$$

2.14 (a) Perl if statements (from perl.com)

$$\begin{aligned} \mathit{IfStatement} \rightarrow\ & \texttt{if}\ \texttt{(}\ \mathit{Expression}\ \texttt{)}\ \mathit{BLOCK} \\ & \{\ \texttt{elsif}\ \texttt{(}\ \mathit{Expression}\ \texttt{)}\ \mathit{BLOCK}\ \} \\ & [\ \texttt{else}\ \mathit{BLOCK}\ ] \end{aligned}$$

Here, *BLOCK* is a sequence of statements (enclosed in curly braces).

(b) Python if statements (from python.org)

$$\begin{aligned} \mathit{IfStatement} \rightarrow\ & \texttt{if}\ \mathit{Expression}\ \texttt{:}\ \ \mathit{BLOCK} \\ & \{\ \texttt{elif}\ \mathit{Expression}\ \texttt{:}\ \ \mathit{BLOCK}\ \} \\ & [\ \texttt{else}\ \mathit{BLOCK}\ ] \end{aligned}$$

(c) Ada if statements (from www.adahome.com/rm95)

$$\begin{aligned} \mathit{IfStatement} \rightarrow\ & \texttt{if}\ \mathit{Expression}\ \texttt{then}\ \mathit{BLOCK} \\ & \{\ \texttt{elsif}\ \mathit{Expression}\ \texttt{then}\ \mathit{BLOCK}\ \} \\ & [\ \texttt{else}\ \mathit{BLOCK}\ ] \\ & \texttt{end if;} \end{aligned}$$

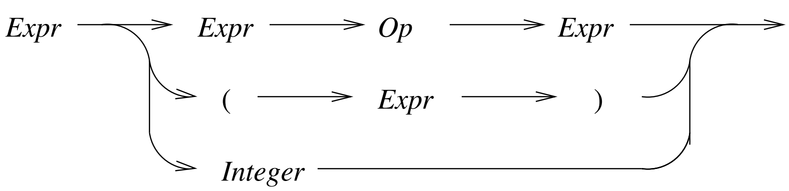
2.15 The rule $a \rightarrow b\ \{\ c\ \}$ is equivalent to:

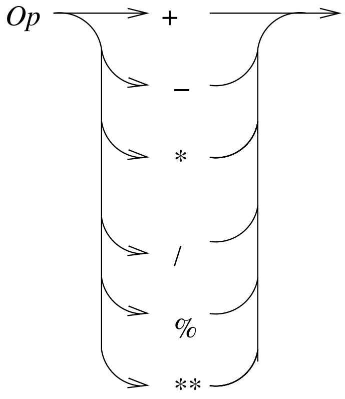
$$\begin{aligned} a &\rightarrow b \mid b\ d \\ d &\rightarrow c \mid d\ c \end{aligned}$$

The rule $a \rightarrow b\ [\ c\ ]$ is equivalent to:

$$a \rightarrow b \mid b\ c$$

2.16 The diagram below omits the syntax of *Integer*.





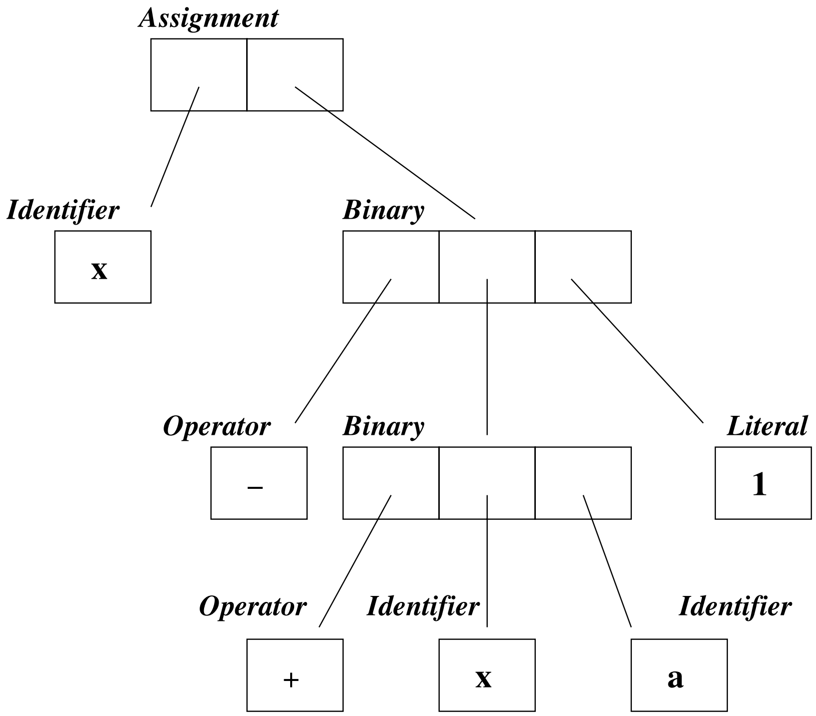
2.17 This should be an essay question if it is answered carefully.

2.18 This type of information can usually be found in an on-line language reference for each language. The number of reserved words we found for each language is: Python 28, Ada95 69, C 32, C++ 67, and Java 50. Because Perl prefixes identifiers with a symbol such as $, @, %, or &, there are no reserved words in Perl. Except for subroutine calls, Perl does not permit "bareword" identifiers.

2.19 See Section 12.7 for an introduction to Perl. In particular, see the program in Figure 12.13 for an example of a variable declaration, which you can use to define a grammar rule like the following:

$$variableDeclaration \rightarrow \texttt{my}\ variableName\ [\ \texttt{=}\ value\ ]\ \texttt{;}$$

2.20 The solution to part (a) appears below. The solutions to parts (b) and (c) are simple variations of this one.


Assignment
Identifier
x
Binary
Operator
–
Binary
Literal
1
Operator
+
Identifier
x
Identifier
a