EXERCISES   Tucker & Noonan [T]

2.1 Using the grammar $G_{integer}$, develop a leftmost derivation for the integer 4520. How many steps are required for this derivation? In general, how many steps are required to derive an integer with an arbitrary number, say d, of *Digits*?

2.2 Using the grammar $G_{integer}$, develop a rightmost derivation for the integer 4520.

2.3 Develop a leftmost derivation for the *Identifier* value *a2i*, using the BNF syntax given in Figure 2.7.

2.4 Develop a rightmost derivation for the *Identifier* value *a2i*, using the BNF syntax given in Figure 2.7.

2.5 Using the grammar of Figure 2.7, draw parse trees for each of the following:

(a) `x = x + a - 1;`
(b) `a = b * c / d;`
(c) `i = i + j * k - 3;`

2.6 Using the following grammar:

$$\begin{aligned} Expr &\rightarrow Expr + Term \mid Expr * Term \mid Term \\ Term &\rightarrow 0 \mid \ldots \mid 9 \mid (\, Expr\,) \end{aligned}$$

draw a parse tree for each of the following:

(a) `5 + 4 * 3`
(b) `5 * 4 + 3`

2.7 Using the following grammar:

$$\begin{aligned} Expr &\rightarrow Term + Expr \mid Term * Expr \mid Term \\ Term &\rightarrow 0 \mid \ldots \mid 9 \mid (\, Expr\,) \end{aligned}$$

draw a parse tree for each of the following:

(a) `5 + 4 * 3`
(b) `5 * 4 + 3`

2.8 Using the following grammar:

$$\begin{aligned} Expr &\rightarrow Expr + Term \mid Term \\ Term &\rightarrow Term \boxed{+} Factor \mid Factor \qquad \ast \\ Factor &\rightarrow 0 \mid \ldots \mid 9 \mid (\, Expr\,) \end{aligned}$$

draw a parse tree for each of the following:

(a) `5 + 4 * 3`
(b) `5 * 4 + 3`

2.9 Using the following grammar:

$$\begin{aligned} Expr \rightarrow\ & Expr + Expr \mid Expr * Expr \mid \\ & 0 \mid \ldots \mid 9 \mid (\, Expr\,) \end{aligned}$$

draw a parse tree for each of the following:

(a) `5 + 4 * 3`
(b) `5 * 4 + 3`

2.10 Using the following grammar:

$$\begin{aligned} \mathit{Expr} \rightarrow\ & + \mathit{Expr}\ \mathit{Expr} \mid * \mathit{Expr}\ \mathit{Expr} \mid \\ & 0 \mid \ldots \mid 9 \end{aligned}$$

draw a parse tree for each of the following:

(a) `+ 5 * 4 3`

(b) `+ * 5 4 3`

2.11 Argue convincingly that the parse tree given in Figure 2.2 for the expression $5 - 4 + 3$ is the only possible parse tree. Hint: enumerate the other possibilities and show that they do not work.

2.12 Show how the Java subgrammar for *IfThenStatement* eliminates any ambiguity in the `if` statement. That is, sketch the parse tree using the Java BNF rules, and then argue that no other parse trees can be found using these rules.

2.13 Consider the grammar rules for *IfStatement* and *Statement* given in Section 2.1.5. Show how they can be altered to eliminate their ambiguity illustrated in the text. That is, show how the grammar can be changed so that the programmer can explicitly distinguish the two alternative attachments of the "dangling else," as shown below:

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

Here, the new keyword `fi` is used to end any statement that begins with the keyword `if`. (It functions like a right brace, where the `if` is the left brace.)

2.14 Give a grammar and an example `if` statement for each of the following languages: (a) Perl, (b) Python, (c) Ada.

2.15 Give translation rules for the EBNF alternatives involving metabrackets and metaparentheses to standard BNF.

2.16 Rewrite grammar G_2 as a set of syntax diagrams.

2.17 Pick one of the following languages: Perl, Python, Ada, C, C++, Java, or another instructor-approved language. Consult an authoritative source and write a report that summarizes the definition of the language with respect to size and type of its grammar, the appearance of ambiguity in the grammar, its number of reserved words, the precedence and associativity of operators, and so on. Cite your sources.

2.18 Research the number of reserved words in each of the following languages: (a) Perl, (b) Python, (c) Ada, (d) C, (e) C++, (f) Java.

2.19 Give a set of grammar rules that define the syntax of a variable declaration in Perl. Give an example.

2.20 Draw an abstract syntax tree for each of the parse trees that you defined for Exercise 2.5, using the abstract syntax for *Assignment* given in Figure 2.11.