Ch 1

3. Pros:
   - Reduces the costs of programming training, compiler purchase, and maintenance;
   - It would simplify programmer recruiting and justify the development of numerous language dependent software development aids.

4. Cons:
   - The language would necessarily be huge and complex;
   - Compilers would be expensive and costly to maintain;
   - The language would probably not be very good for any programming domain, either in compiler efficiency or in the efficiency of the code it generated.
   - More importantly, it would not be easy to use, because regardless of the application area, the language would include many unnecessary and confusing features and constructs (those meant for other application areas).
   - Different users would learn different subsets, making maintenance difficult.

7. Pro:
   - simplicity—a right brace always terminates a compound.

Con:
   - when you see a right brace in a program, the location of its matching left brace is not always obvious, in part because all multiple-statement control constructs end with a right brace.

9. cf. Slide 13

18. The main disadvantage of using paired delimiters for comments is that it results in diminished reliability. It is easy to inadvertently leave off the final delimiter, which extends the comment to the end of the next
comment, effectively removing code from the program. The advantage of paired delimiters is that you *can* comment out areas of a program. The disadvantage of using only beginning delimiters is that they must be repeated on every line of a block of comments. This can be tedious and therefore error-prone. The advantage is that you cannot make the mistake of forgetting the closing delimiter.

**Ch 2**

6. Because of the simple syntax of LISP, few syntax errors occur in LISP programs. Unmatched parentheses is the most common mistake.

7. Execution efficiency.

10. The main motivation for the development of PL/I was to provide a single tool for computer centers that must support both scientific and commercial applications. IBM believed that the needs of the two classes of applications were merging, at least to some degree. They felt that the simplest solution for a provider of systems, both hardware and software, was to furnish a single hardware system running a single programming language that served both scientific and commercial applications.

11. IBM was, for the most part, incorrect in its view of the future of the uses of computers, at least as far as languages are concerned. Commercial applications are nearly all done in languages that are specifically designed for them. Likewise for scientific applications. On the other hand, the IBM design of the 360 line of computers was a great success--it still dominates the area of computers between supercomputers and minicomputers. Furthermore, 360 series computers and their descendants have been widely used for both scientific and commercial applications. These applications have been done, in large part, in Fortran and COBOL.

12. Slide 10 may help.
14. The argument for typeless languages is their great flexibility for the programmer. Literally, any storage location can be used to store any type value. This is useful for very low-level languages used for systems programming. The drawback is that type checking is impossible, so that it is entirely the programmer's responsibility to insure that expressions and assignments are correct.

Ch 3

2a. `<class_head> → {<modifier>} class <id> [extends class_name] [implements <interface_name> {, <interface_name>]}
   <modifier> → public | abstract | final

2c. `<switch_stmt> → switch (<expr>) {case <literal> : <stmt_list>
   {case <literal> : <stmt_list> } [default : <stmt_list> ]}

3. `<assign> → <id> = <expr>
   <id> → A | B | C
   <expr> → <expr> * <term>
   | <term>
   <term> → <factor> + <term>
   | <factor>
   <factor> → ( <expr> )
   | <id>

4. `<assign> → <id>++ | <id>-- | ++<id> | --<id>

6.
(a) `<assign> => <id> = <expr>
    => A = <expr>
    => A = <id> * <expr>
    => A = A * <expr>
8. The following two distinct parse trees for the same string prove that the grammar is ambiguous.
9. Assume that the unary operators can precede any operand. Replace the rule
\[ \text{factor} \rightarrow \text{id} \]
with
\[ \text{factor} \rightarrow + \text{id} \]
\[ \quad \mid - \text{id} \]

10. One or more a's followed by one or more b's followed by one or more c's.
    e.g. \textit{aabccccc} and \textit{abc} are in the language but \textit{aab} or \textit{aaccbb} are not.

12. a and e

13. \( S \rightarrow a \ S \ b \mid a \ b \)

16. \( \text{assign} \rightarrow \text{id} = \text{expr} \)
    \[ \text{id} \rightarrow A \mid B \mid C \]
    \[ \text{expr} \rightarrow \text{expr} (+ \mid -) \text{expr} \]
    \[ \quad \mid (\text{expr}) \]
    \[ \quad \mid \text{id} \]
18. (a) (Pascal \texttt{repeat-until}) We assume that the logic expression is a single relational expression.

\begin{verbatim}
loop: (repeat body)
  if \texttt{<relational_expression>} goto out
  goto loop
out: ...
\end{verbatim}

(b) (Ada \texttt{for}) \texttt{for I in first .. last loop}

\begin{verbatim}
I = first
loop: if I < last goto out
...
  I = I + 1
goto loop
out: ...
\end{verbatim}

(c) (Fortran \texttt{Do})

\begin{verbatim}
K = start
loop: if K > end goto out
...
  K = K + step
goto loop
out: ...
\end{verbatim}

(d)

(e) (C \texttt{for}) \texttt{for (expr1; expr2; expr3) ...}

\begin{verbatim}
evaluate(expr1)
loop: control = evaluate(expr2)
\end{verbatim}
if control == 0 goto out
...
evaluate(expr3)
goto loop
out: ...

(f)

19.
(a) \( a = 2 \times (b - 1) - 1 \) \{a > 0\}
    \( 2 \times (b - 1) - 1 > 0 \)
    \( 2 \times b - 2 - 1 > 0 \)
    \( 2 \times b > 3 \)
    \( b > 3 / 2 \)

(b) \( b = (c + 10) / 3 \) \{b > 6\}
    \( (c + 10) / 3 > 6 \)
    \( c + 10 > 18 \)
    \( c > 8 \)

(c) \( a = a + 2 \times b - 1 \) \{a > 1\}
    \( a + 2 \times b - 1 > 1 \)
    \( 2 \times b > 2 - a \)
    \( b > 1 - a / 2 \)

(d) \( x = 2 \times y + x - 1 \) \{x > 11\}
    \( 2 \times y + x - 1 > 11 \)
    \( 2 \times y + x > 12 \)

20.
(a) \( a = 2 \times b + 1 \)
    \( b = a - 3 \) \{b < 0\}
    \( a - 3 < 0 \)
a < 3

Now, we have:
\[
\begin{align*}
  a &= 2 \times b + 1 \quad \{a < 3\} \\
  2 \times b + 1 &< 3 \\
  2 \times b + 1 &< 3 \\
  2 \times b &< 2 \\
  b &< 1
\end{align*}
\]

(b) \quad a = 3 \times (2 \times b + a); \\
     b = 2 \times a - 1 \quad \{b > 5\} \\
\[
\begin{align*}
  2 \times a - 1 &> 5 \\
  2 \times a &> 6 \\
  a &> 3
\end{align*}
\]

Now we have:
\[
\begin{align*}
  a &= 3 \times (2 \times b + a) \quad \{a > 3\} \\
  3 \times (2 \times b + a) &> 3 \\
  6 \times b + 3 \times a &> 3 \\
  2 \times b + a &> 1 \\
  n &> (1 - a) / 2
\end{align*}
\]

21a. \(M_{pf}(\text{for var in init_expr .. final_expr loop L end loop, s}) \equiv\)
   if \(\text{VARMAP}(i, s) = \text{undef}\) for var or some i in init_expr or final_expr
   then \text{error}
   else if \(M_e(\text{init_expr, s}) > M_e(\text{final_expr, s})\)
   then s
   else \(M_l(\text{while init_expr - 1 <= final_expr do L, M}_a(\text{var := init_expr + 1, s}))\)

21b. \(M_r(\text{repeat L until B}) \equiv\)
   if \(M_b(B, s) = \text{undef}\)
   then \text{error}
else if $M_{sl}(L, s) = \text{error}$
  then \text{error}
else if $M_{b}(B, s) = \text{true}$
  then $M_{sl}(L, s)$
else $M_{r}$(repeat $L$ until $B$), $M_{sl}(L, s))$

21c. $M_{b}(B, s) \equiv$ if $\text{VARMAP}(i, s) = \text{undef}$ for some $i$ in $B$
  then \text{error}
else $B'$, where $B'$ is the result of
  evaluating $B$ after setting each
  variable $i$ in $B$ to $\text{VARMAP}(i, s)$

21d. $M_{cf}$(for $(\text{expr1}; \text{expr2}; \text{expr3}) L, s) \equiv$
  if $\text{VARMAP} (i, s) = \text{undef}$ for some $i$ in $\text{expr1}$, $\text{expr2}$, $\text{expr3}$, or $L$
  then \text{error}
else if $M_{e} (\text{expr2}, M_{e} (\text{expr1}, s)) = 0$
  then $s$
else $M_{\text{help}} (\text{expr2}, \text{expr3}, L, s)$

$M_{\text{help}} (\text{expr2}, \text{expr3}, L, s) \equiv$
if $\text{VARMAP} (i, s) = \text{undef}$ for some $i$ in $\text{expr2}$, $\text{expr3}$, or $L$
  then \text{error}
else
  if $M_{sl} (L, s) = \text{error}$
    then $s$
  else $M_{\text{help}} (\text{expr2}, \text{expr3}, L, M_{sl} (L, M_{e} (\text{expr3}, s))$

22. The value of an intrinsic attribute is supplied from outside the
attribute evaluation process, usually from the lexical analyzer. A value
of a synthesized attribute is computed by an attribute evaluation
function.