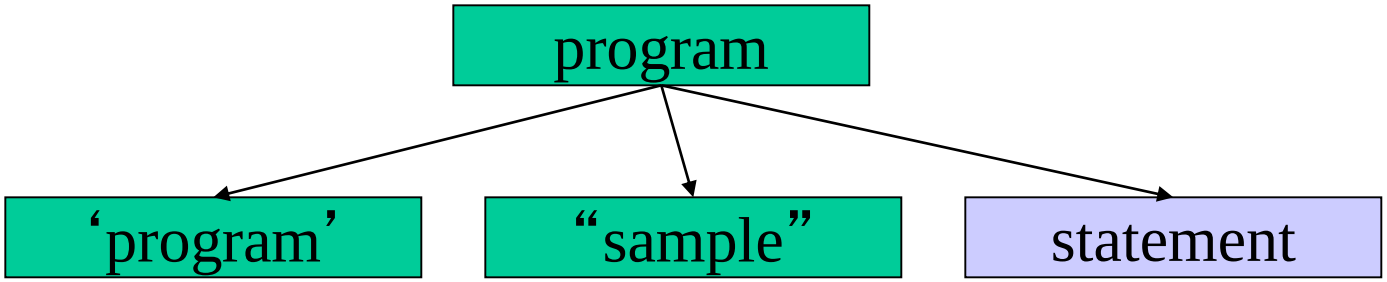
Tutorial exercise 1 – note on solutions

- Grammar:

```
Program -> ‘program’ string statement
Statement -> ‘turn’ number ‘degrees’ |
             ‘forward’ number |
             ‘times’ number ‘do’ statement |
             ‘begin’ statementlist
Statement-list -> ‘end’ |
                  statement ‘;’ statement-list
```

- Stick to a simple rule:
  - Start by drawing the start symbol “program” at the top of the page
  - Draw three arrows, one for each item on the right hand side:
    - ‘program’ string statement

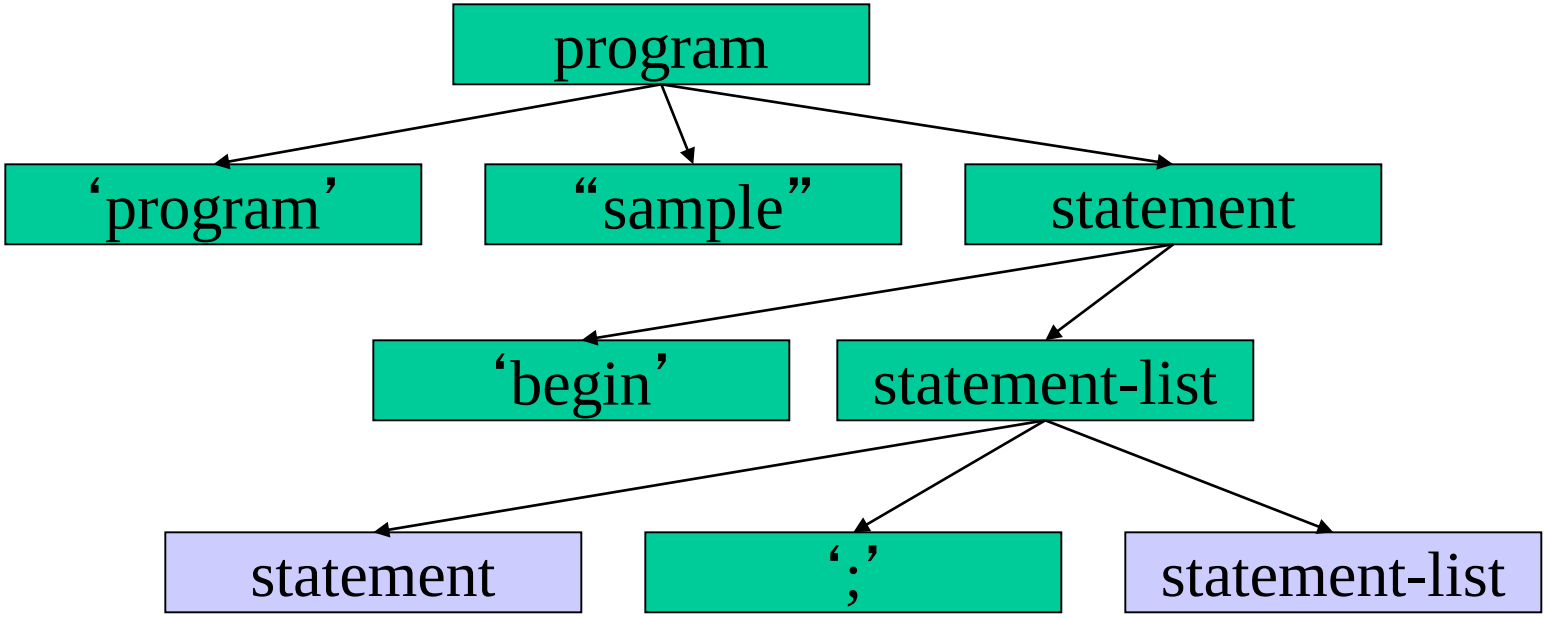
Drawing the parse tree



- Grammar:

```
Program -> ‘program’ string statement
```

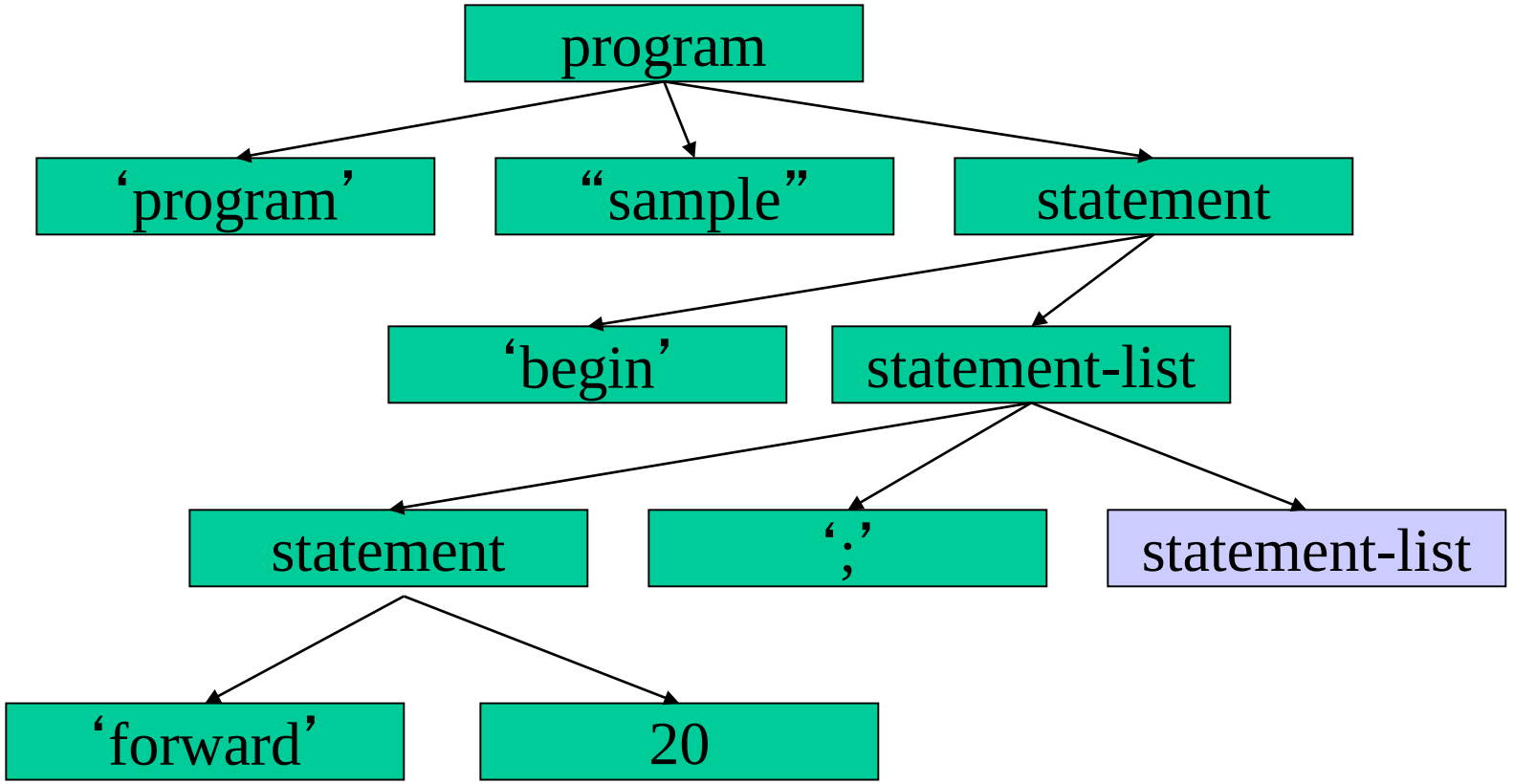
Drawing the parse tree



- Grammar:

```
Statement -> ‘turn’ number ‘degrees’ |
                    ‘forward’ number |
                    ‘times’ number ‘do’ statement |
                    ‘begin’ statementlist
Statement-list -> ‘end’ |
                  statement ‘;’ statement-list
```

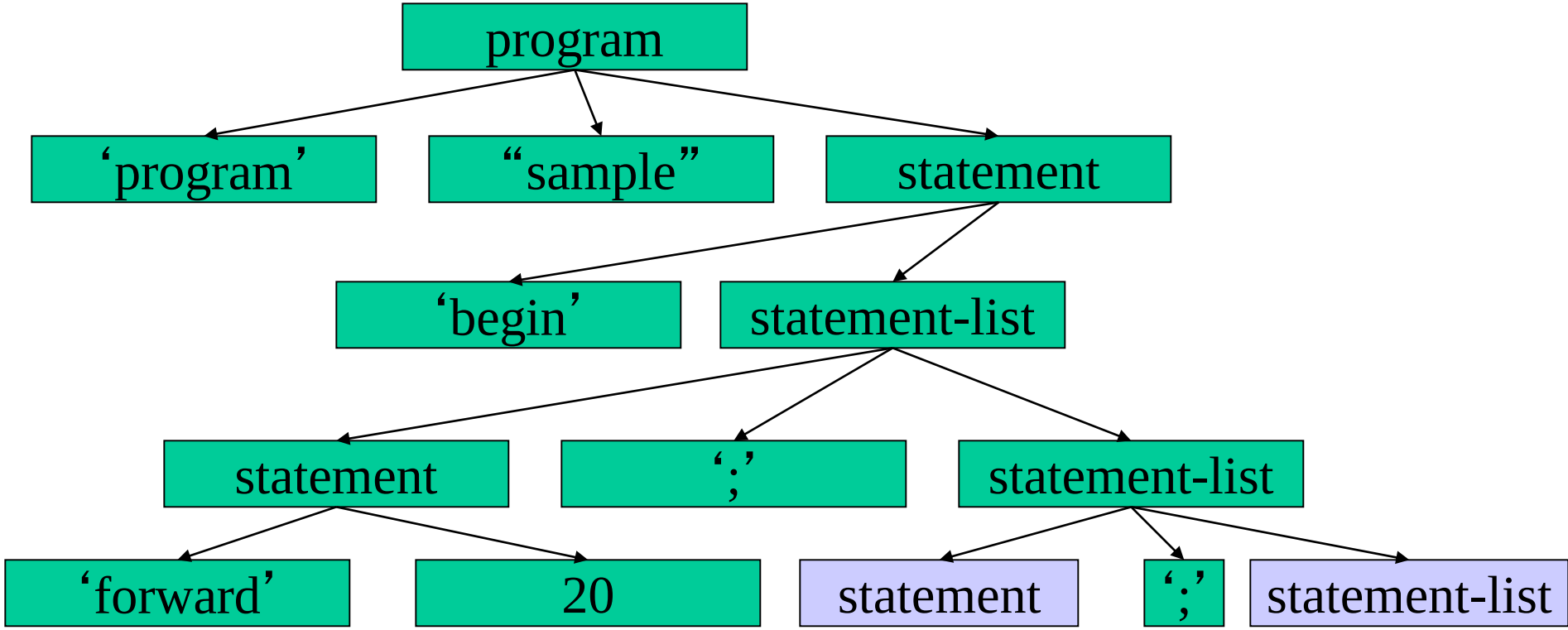
Drawing the parse tree



•Grammar:

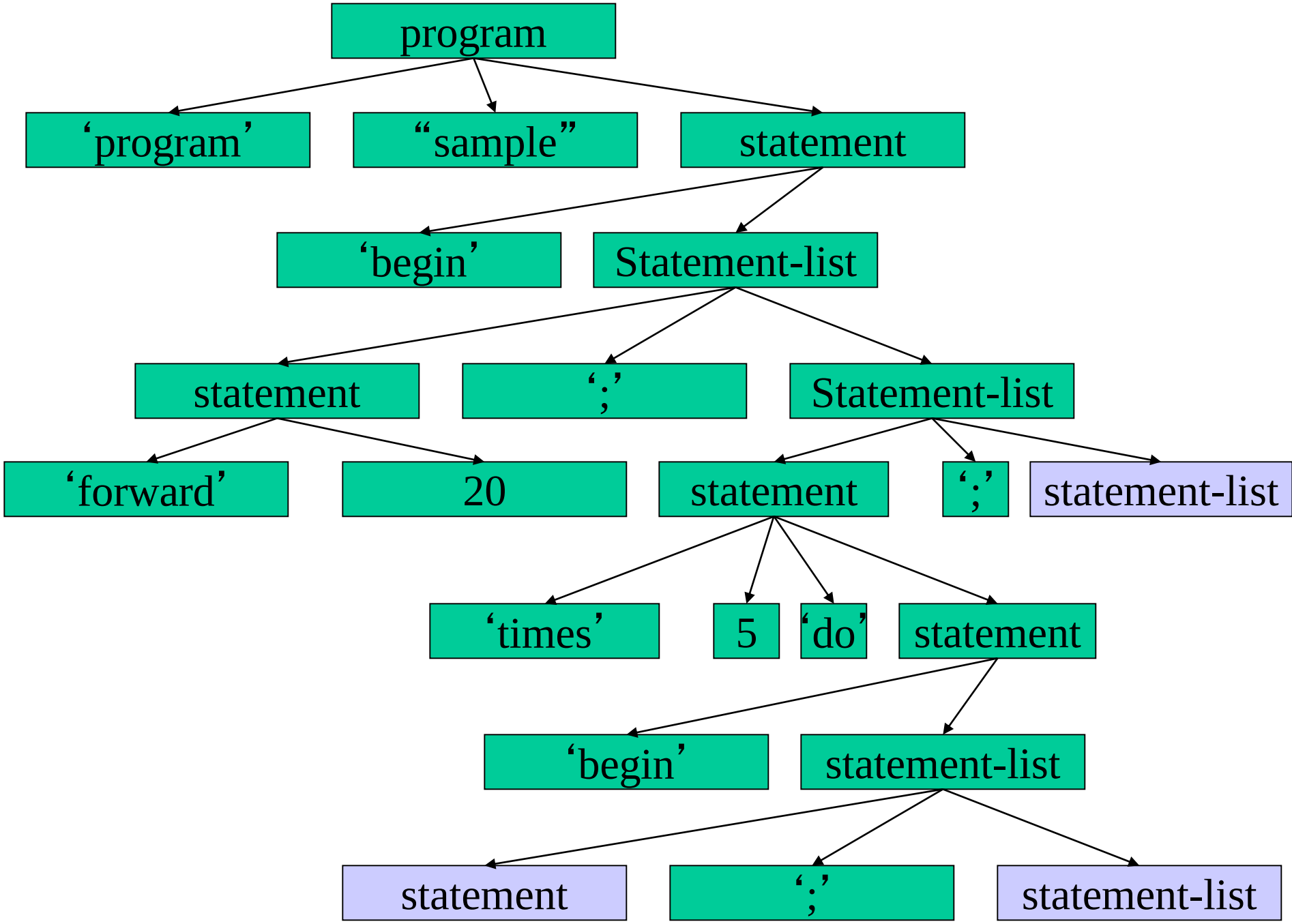
```
Statement -> ‘turn’ number ‘degrees’ |
                        ‘forward’ number | …
                        ‘times’ number ‘do’ statement |
                        ‘begin’ statementlist
Statement-list -> ‘end’ |
                    statement ‘;’ statement-list
```

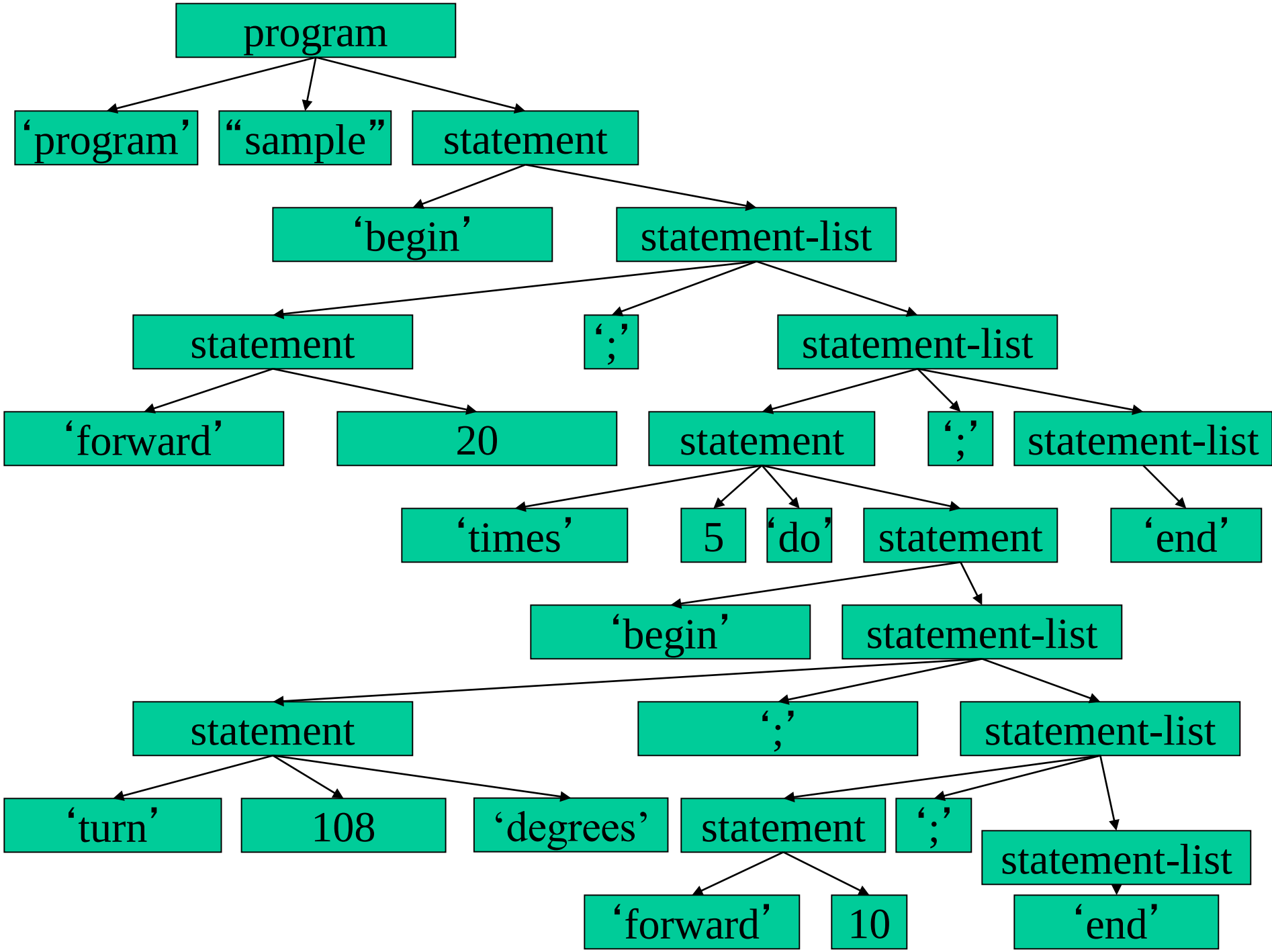
Drawing the parse tree

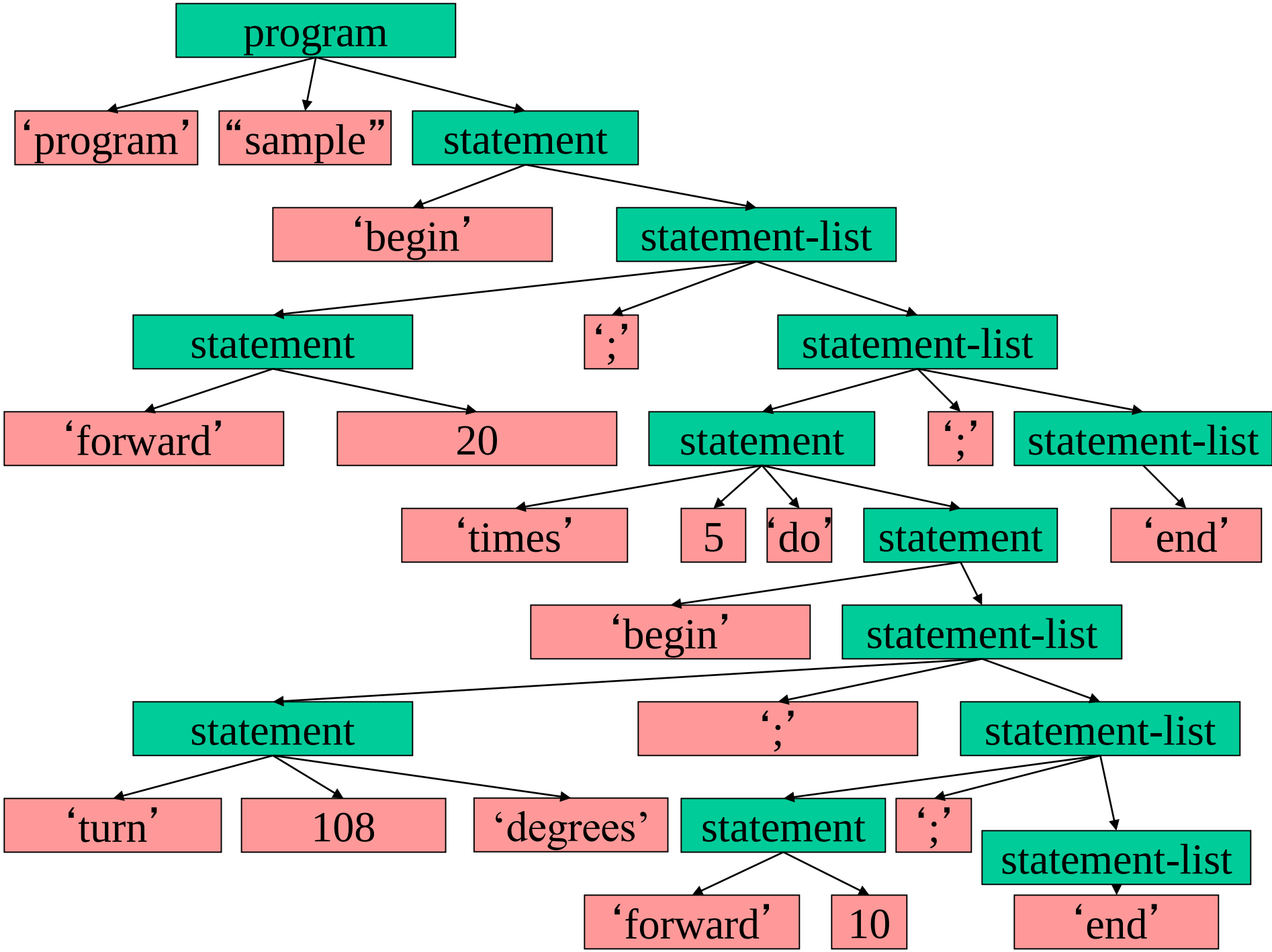


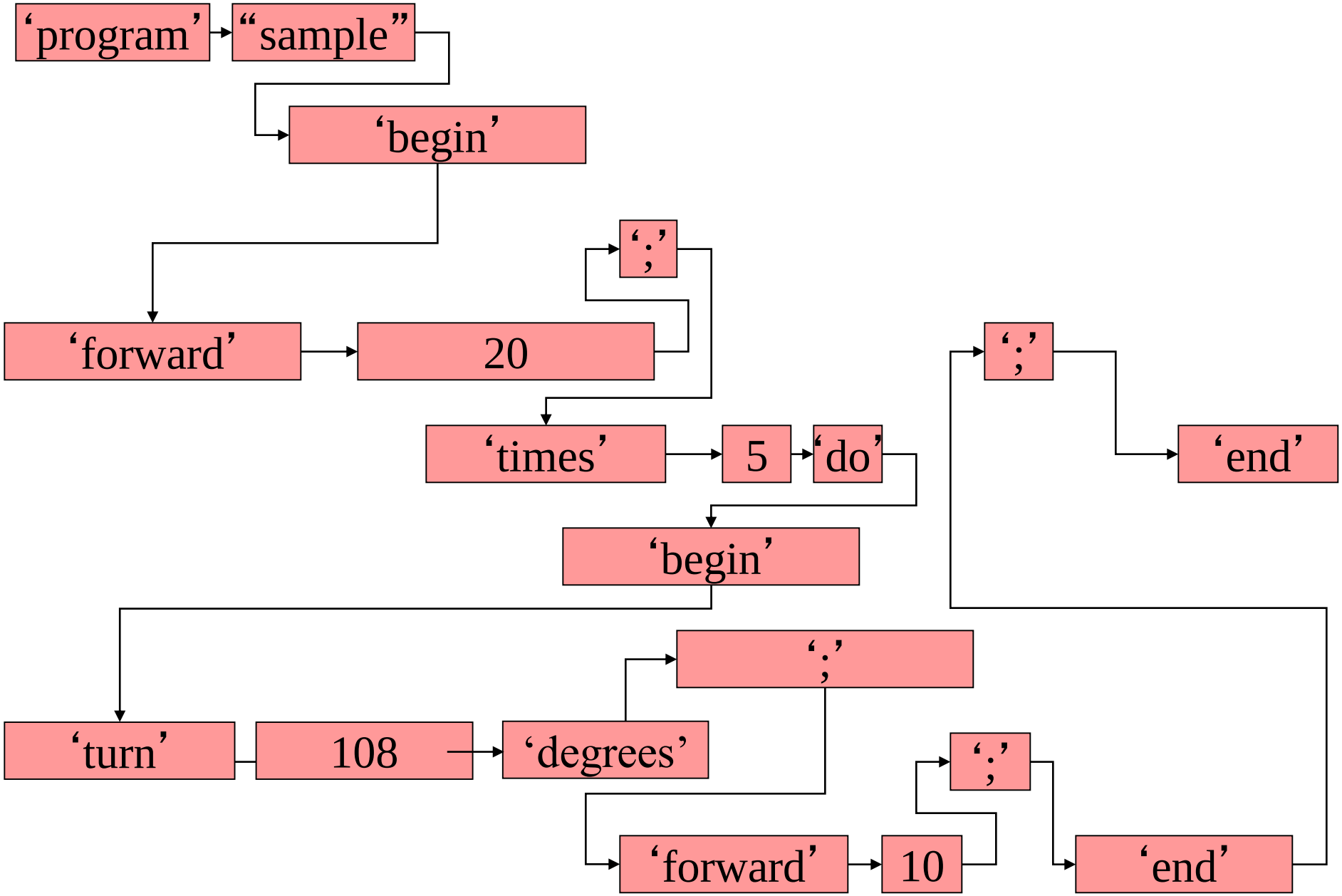
- Grammar:

```
Statement-list -> ‘end’ |
                  statement ‘;’ statement-list
```


program
‘program’
“sample”
statement
‘begin’
Statement-list
statement
‘;’
Statement-list
‘forward’
20
statement
‘;’
statement-list
‘times’
5
‘do’
statement
‘begin’
statement-list
statement
‘;’
statement-list


program
‘program’
“sample”
statement
‘begin’
statement-list
statement
‘;’
statement-list
‘forward’
20
statement
‘;’
statement-list
‘times’
5
‘do’
statement
‘end’
‘begin’
statement-list
statement
‘;’
statement-list
‘turn’
108
‘degrees’
statement
‘;’
statement-list
‘forward’
10
‘end’


program
‘program’
“sample”
statement
‘begin’
statement-list
statement
‘;’
statement-list
‘forward’
20
statement
‘;’
statement-list
‘times’
5
‘do’
statement
‘end’
‘begin’
statement-list
statement
‘;’
statement-list
‘turn’
108
‘degrees’
statement
‘;’
statement-list
‘forward’
10
‘end’


‘program’
“sample”
‘begin’
‘forward’
20
‘;’
‘times’
5
‘do’
‘begin’
‘turn’
108
‘degrees’
‘;’
‘forward’
10
‘;’
‘;’
‘end’
‘end’

Turtle.java

```
static StatementTree parseStatement(Lexer lex) throws IOException
   {
      Token t = lex.nextToken();
      switch (t.tokenId) {
      case Token.TURN:
          lex.match(Token.NUMBER);
          int degrees = lex.getLastToken().intValue;
          lex.match(Token.DEGREES);
          return new TurnNode(degrees);

      case Token.FORWARD:
```

??

??

??

Turtle.java

```
static StatementTree parseStatement(Lexer lex) throws IOException
    {
        Token t = lex.nextToken();
        switch (t.tokenId) {
        case Token.TURN:
            lex.match(Token.NUMBER);
            int degrees = lex.getLastToken().intValue;
            lex.match(Token.DEGREES);
            return new TurnNode(degrees);

        case Token.FORWARD:
            lex.match(Token.NUMBER);
            int distance = lex.getLastToken().intValue;
            return new ForwardNode(distance);
```

Turtle.java

```
static StatementTree parseStatement(Lexer lex) throws IOException
  {
    Token t = lex.nextToken();
    switch (t.tokenId) {
    case Token.TURN:
      …
    case Token.FORWARD:
      …
    case Token.TIMES:
      ??
      ??
      ??
      ??
      ??
```

Turtle.java

```
static StatementTree parseStatement(Lexer lex) throws IOException
  {
    Token t = lex.nextToken();
    switch (t.tokenId) {
    case Token.TURN:
      …
    case Token.FORWARD:
      …
    case Token.TIMES:
      lex.match(Token.NUMBER);
      int count = lex.getLastToken().intValue;
      lex.match(Token.DO);
      ??
      ??
```

Turtle.java

```
static StatementTree parseStatement(Lexer lex) throws IOException
  {
    Token t = lex.nextToken();
    switch (t.tokenId) {
    case Token.TURN:
      …
    case Token.FORWARD:
      …
    case Token.TIMES:
      lex.match(Token.NUMBER);
      int count = lex.getLastToken().intValue;
      lex.match(Token.DO);
      StatementTree body = ??
      return new TimesNode(count, body);
```

Turtle.java

```
static StatementTree parseStatement(Lexer lex) throws IOException
  {
    Token t = lex.nextToken();
    switch (t.tokenId) {
    case Token.TURN:
      …
    case Token.FORWARD:
      …
    case Token.TIMES:
      lex.match(Token.NUMBER);
      int count = lex.getLastToken().intValue;
      lex.match(Token.DO);
      StatementTree body = parseStatement(lex);
      return new TimesNode(count, body);
```

InterpretVisitor.java

```
public class InterpretVisitor extends TreeVisitor {
   void visitStatementList(StatementTree first,
                           StatementTreeList rest) {
      first.Accept(this);
      if (rest != null) {
         rest.Accept(this);
      }
   }
   void visitTurnNode(int degrees) {
      System.out.println("Please turn "+degrees+" degrees");
   }
   void visitForwardNode(int distance) {
      ??
   }
   void visitTimesNode(int count, StatementTree body) {
      ??
      ??
      ??
   }
   void visitBeginNode(StatementTreeList body) {
      body.Accept(this);
   }
}
```

InterpretVisitor.java

```
public class InterpretVisitor extends TreeVisitor {
   void visitStatementList(StatementTree first,
                           StatementTreeList rest) {
      first.Accept(this);
      if (rest != null) {
         rest.Accept(this);
      }
   }
   void visitTurnNode(int degrees) {
      System.out.println("Please turn "+degrees+" degrees");
   }
   void visitForwardNode(int distance) {
      System.out.println("Please move forward "+distance);
   }
   void visitTimesNode(int count, StatementTree body) {
      ??
      ??
      ??
   }
   void visitBeginNode(StatementTreeList body) {
      body.Accept(this);
   }
}
```

InterpretVisitor.java

```
public class InterpretVisitor extends TreeVisitor {
   void visitStatementList(StatementTree first,
                           StatementTreeList rest) {
      first.Accept(this);
      if (rest != null) {
         rest.Accept(this);
      }
   }
   void visitTurnNode(int degrees) {
      System.out.println("Please turn "+degrees+" degrees");
   }
   void visitForwardNode(int distance) {
      System.out.println("Please move forward "+distance);
   }
   void visitTimesNode(int count, StatementTree body) {
      for (int i=0; i<count; ++i) {
         body.Accept(this);
      }
   }
   void visitBeginNode(StatementTreeList body) {
      body.Accept(this);
   }
}
```

Visitors

- The turtle interpreter was implemented using a “visitor”
- Visitor is an example of a “design pattern”
- Visitor is a common technique to simplify traversal of a tree or graph
- What is the alternative?

If you don’t use a visitor…

```
public class TurnNode extends StatementTree {
    int degrees;

    TurnNode(int d) {
        degrees = d;
    }
    public void print() {
        System.out.println("turn "+degrees+" degrees");
    }
    public void interpret() {
        System.out.println("please turn "+degrees);
    }

}
```

- The simplest way to implement the interpreter is to have an “interpret()” method for each of the AST’s node types – as shown above for “TurnNode”

If you don’t use a visitor…

```
public class TurnNode extends StatementTree {
    int degrees;

    TurnNode(int d) {
        degrees = d;
    }
    public void print() {
        System.out.println("turn "+degrees+" degrees");
    }
    public void interpret() {
        System.out.println("please turn "+degrees);
    }
    public void inFrench() {
        System.out.println(“tournez "+degrees);
    }
}
```

- We need to add a method for each operation that involves a traversal of the AST

Using a visitor…

```
public class TurnNode extends StatementTree {
    int degrees;

    TurnNode(int d) {
        degrees = d;
    }
public void Accept(TreeVisitor v) {
        v.visitTurnNode(degrees);
    }
}
```

- With a visitor the AST node types just have one Accept method

```
public class InterpretVisitor extends TreeVisitor {
   void visitStatementList(StatementTree first,
                           StatementTreeList rest) {
      first.Accept(this);
      if (rest != null) {
         rest.Accept(this);
      }
   }
   void visitTurnNode(int degrees) {
      System.out.println("Please turn "+degrees+" degrees");
   }
   void visitForwardNode(int distance) {
      System.out.println("Please move forward "+distance);
   }
   void visitTimesNode(int count, StatementTree body) {
      for (int i=0; i<count; ++i) {
         body.Accept(this);
      }
   }
   void visitBeginNode(StatementTreeList body) {
      body.Accept(this);
   }
}
```

- Now we can encapsulate all the interpreter code in a single file
- And we can write a “print” traversal in a similar, single file