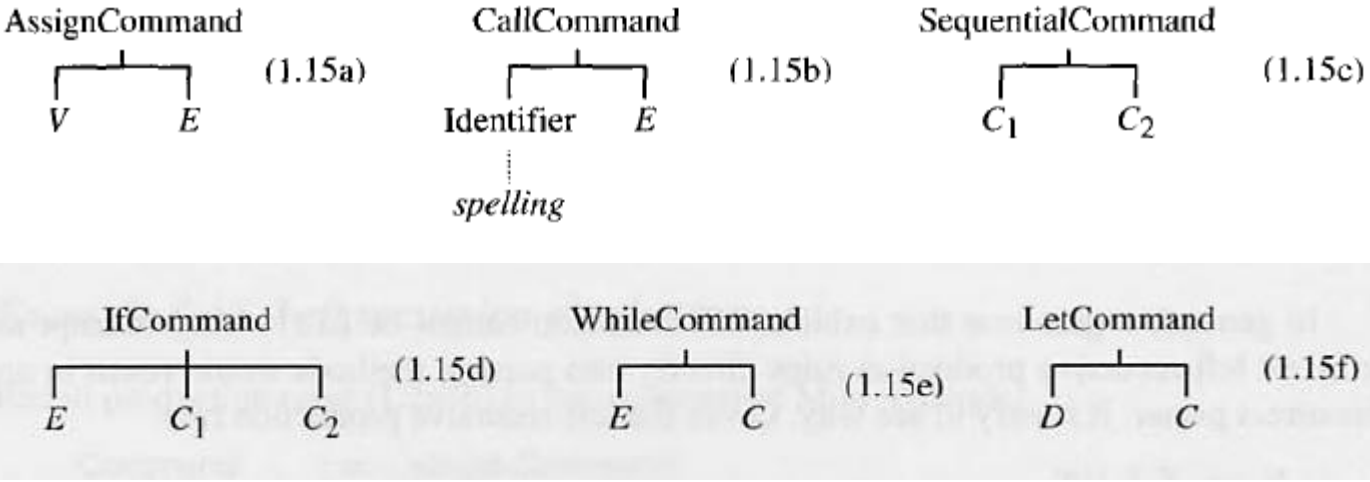
AssignCommand
V
E
(1.15a)
CallCommand
Identifier
E
spelling
(1.15b)
SequentialCommand
C_1
C_2
(1.15c)
IfCommand
E
C_1
C_2
(1.15d)
WhileCommand
E
C
(1.15e)
LetCommand
D
C
(1.15f)

```
public abstract class AST {
    ...
}

public abstract class Command extends AST { ... }

public class AssignCommand extends Command {
    public Vname V;                    // left-side variable
    public Expression E;               // right-side expression
    ...
}

public class CallCommand extends Command {
    public Identifier I;               // procedure name
    public Expression E;               // actual parameter
    ...
}

public class SequentialCommand extends Command {
    public Command C1, C2;             // subcommands
    ...
}
```

```
public class IfCommand extends Command {
    public Expression E;          // if condition
    public Command C1, C2;        // true and false commands
    ...
}
public class WhileCommand extends Command {
    public Expression E;          // loop condition
    public Command C;             // body of loop
    ...
}
public class LetCommand extends Command {
    public Declaration D;         // block declarations
    public Command C;             // body of block
    ...
}
```

```
private ASTN parseN () {
    ASTN itsAST;
    parse X, at the same time constructing itsAST
    return itsAST;
}
```

```
private Program         parseProgram ();
private Command         parseCommand ();
private Command         parseSingleCommand ();
private Expression      parseExpression ();
private Expression      parsePrimaryExpression ();
private Declaration     parseDeclaration ();
private Declaration     parseSingleDeclaration ();
private TypeDenoter     parseTypeDenoter ();
private Identifier      parseIdentifier ();
private IntegerLiteral  parseIntegerLiteral ();
private Operator        parseOperator ();
```

```
private Command parseSingleCommand () {
    Command comAST;
    switch (currentToken.kind) {

    case Token.IDENTIFIER: {
            Identifier iAST = parseIdentifier();
            switch (currentToken.kind) {
            case Token.BECOMES: {
                    acceptIt();
                    Expression eAST = parseExpression();
                    comAST = new AssignCommand(iAST, eAST);
                }
                break;
            case Token.LPAREN: {
                    acceptIt();
                    Expression eAST = parseExpression();
                    accept(Token.RPAREN);
                    comAST = new CallCommand(iAST, eAST);
                }
                break;
```

```
            default:
                report a syntactic error
            }
        }
        break;

    case Token.IF:
        ...
    case Token.WHILE:
        ...
    case Token.LET: {
            acceptIt();
            Declaration dAST = parseDeclaration();
            accept(Token.IN);
            Command cAST = parseSingleCommand();
            comAST = new LetCommand(dAST, cAST);
        }
        break;

    case Token.BEGIN: {
            acceptIt();
            comAST = parseCommand();
            accept(Token.END);
        }
        break;

    default:
        report a syntactic error
    }
    return comAST;
}
```

```
public class Parser {
    private Token currentToken;
    ...  // Auxiliary methods.
    ...  // Enhanced parsing methods, as above.
    public Program parse () {
        currentToken = scanner.scan();
        Program progAST = parseProgram();
        if (currentToken.kind != Token.EOT)
            report a syntactic error
        return progAST;
    }
}
```

PROJETO

Etapa 3: MONTAGEM DA AST

- Construir uma estrutura de dados que represente a estrutura sintática do programa-fonte (AST); para isso, deverão ser especificadas as classes abstratas e concretas que serão utilizadas para representar os nós da árvore. Depois, os métodos de análise sintática deverão ser adaptados para construir a árvore durante o fluxo de processamento do programa-fonte.
- O programa deverá prever uma opção que permita ao usuário visualizar a árvore depois de montada. Utilizar o padrão de projeto VISITOR.