情報科学演習 D
課題 2

基礎工学部情報科学科ソフトウェア科学コース
学籍番号: 09B20033

城間大幹

2022 年 11 月 13 日

1　仕様

本章では Parser[*1]の仕様について説明する．Parser への入力は Pascal 風プログラムを字句解析した結果 (ts ファイル) であり，出力は構文的な誤りが含まれるかどうかの判定結果である．開発対象となる構文解析器のメソッドは Parser.run(String) で ありその仕様は以下の通りである．

- 第一引数で指定された ts ファイルを読み込み，構文解析を行う
- 構文が正しい場合は標準出力に”OK”を出力する
- 構文が正しくない場合は，”Syntax error: line”という文字列とともに，最初の誤りを見つけた行の番号を標準エラーに出力する
- 入力ファイル内に複数の誤りが含まれる場合は，最初に見つけた誤りのみを出力する
- 入力ファイルが見つからない場合は標準エラーに”File not found”と出力して終了する

なおここでは意味的な誤り, 例えばスコープ範囲外の変数を参照した, そもそも定義されていない変数を使用したなどについては無視する．

2　方針と設計

基本的には ts ファイルを 1 行ずつ読み込み, トークン名ごとに指導書にある構文定義を満たすかを確認する．確認方法としては, write 文, if 文, while 文などを判定する関数をそれぞれのトークン名を読み込んだタイミングで呼び出す．また, 条件文や計算式, 括弧が前後揃っているかなどの判定も関数で行う．区切り文字や終端文字の有無などはそれぞれの関数内で確認する．

3　実装プログラム

ここでは Parser を構成する関数の機能について説明する．なお作成した関数は基本的に引数として scanner を持つが, これは常に ts ファイルの最新の行を読み出すためである．また関数は boolean 型であるが, 条件を満たした場合のみ true で終了し, その他 false で終了する場合は全て構文誤りである．

3.0.1　関数 SVAR

関数宣言時, var を読み込んだ際に呼び出される．その後変数名を読み込んだ際, 関数 SIDENTIFIER を呼び出す．

3.0.2　関数 SIDENTIFIER

識別子を読み込んだ場合に呼び出す．主に変数宣言時の var 以降と変数への代入の際に用いる．変数宣言の際には, まずコンマを用いて 1 行でいくつかの変数を同時に宣言する場合は, flag を用いて構文判定を行う．前括弧がくる場合は, 関数 Brackets を呼び出し構文判定を行う．そしてコロンの後の変数のタイプが構文定義を満たすかを判定する．

[*1] 今回作成したプログラムの名前

代入の際は, コロンイコール後に演算なしで代入するものはそのまま, 演算ありで代入するのものは関数 calculation を呼び出して構文判定を行う.

3.1 関数 calculation

演算子と被演算子を処理していく関数であるため, 基本的にはこれら以外のものが現れた時点で関数を終了する. それ故に, SCONSTANT や SIDENTIFIER のような被演算子や, SPLUS や SMINUS の演算子を読み取った場合のみ次の行を読み込む. また関数 calculation で左括弧を読み込んだ場合, 関数 Brackets が呼び出される. なおここで用いられている flag は 関数 Brackets を実行した後のセミコロンを読むためであり, ブロック的な役割を持つ.

3.2 関数 Brackets

左括弧を読み出した時に呼び出す関数であり, 括弧に関してさまざまな用途で用いる. 例えば関数 Brackets の前半部では副プログラム文の定義時や呼び出し時の引数の構文判定を行なっている. また演算に関わる場合は, 演算子を読み出したタイミングで関数 calculation を呼び出している. そして関数 conditionalExpression と同様に, セミコロンや比較演算子, STHEN, SDO で関数を抜ける. なおここで用いられている flag はコンマで変数一つ一つを区切って表す時の構文判定のため, something は無限ループを防ぐためのものである.

3.3 関数 conditionalExpression

if 文や while 文の条件式の構文判定で用いる関数である. 条件式では単行演算子と 比較演算子を用いる場合の 2 種類存在する. したがってまず関数 calculation を用いて識別子などを処理する. そして単行演算子である場合 (ex. boolean 型の変数 1 つのみの条件式) は, 次に SDO や STHEN を読み取り関数を終了する. 比較演算子があった場合はその後関数 calculation を再度呼び出して右辺を処理し, その後の SDO や STHEN で関数を終了する.

なお関数 conditionalExpression では brflag という変数を用いているがこれは, 括弧の構文判定のために用いている. 演算に関連する括弧は関数 calculation と関数 Brackets との対応で判定しているが, 本質的には意味のない括弧[*2]にはそれで対応できない. したがって, 左括弧で brflag++, 右括弧で brflag−− をすることにより, 関数 conditionalExpression 終了時に brflag が 0 であることで判定する.

3.4 関数 SIF

if 文宣言時, if を読み込んだ際に呼び出す. 条件式は関数 conditionalExpression で判定する. その後, begin-end 内で入れ子の if 文や while 文が呼び出されていないかの判定や, else 文の構文判定などを行う.

3.5 関数 SWHILE

while 文宣言時, while を読み込んだ際に呼び出す. 条件式は関数 conditionalExpression で判定する. その後, begin-end 内で入れ子の if 文や while 文が呼び出されていないかなどを確認して, 構文判定を行う.

3.6 関数 SPROCEDURE

副プログラム宣言時, procedure を読み込んだ際に呼び出す. その後手続き名を読み込み, 引数がある場合は関数 Brackets を呼び出して構文判定を行う.

[*2] 比較演算子を挟んだ項を括弧で挟む. Ex $if(a+b+c) > (d*e)$

3.7 関数 SWRITELN

write 文宣言時, write を読み込んだ際に呼び出す. 基本的には丸括弧が前後存在し, その中に文字列か識別子があるかを見て構文判定する.

3.8 関数 grammerCorrect

ts ファイルを全て読み込み構文的な誤りがなかった際に OK を出力する関数.

3.9 関数 grammerError

構文的な誤りが見つかった場合に, 誤りがあった行数とともにエラーメッセージを表示する関数.

4 考察や工夫点

工夫した点は, 構文木を明確には意識しなかったもののなるべく構文定義をブロック化し, それぞれを関数にしてうまく組み合わせるというやり方である. これにより, 一度クリアしたテストケースが通らなくなったり, 同じ構文でもケースにより判定結果が異なったりするということがほとんどなかった.

ただ演算子, 被演算子, 括弧が複雑に交わる場合はあまり効果的な記述ができなかったので, その点はさらに改善したい.

5 感想

課題 1 では一つのテストケースに依存しすぎないよう, 多くのテストケースで発生する問題から優先的に対処したが, この課題 2 では難易度もあってか, 1 から順番にテストケースを通していくような, テストケース主体のプログラム構築になった. しかし, ブロック化した構文定義を関数にして表すという大枠の方針があったため思いの外スムーズに進んだ. 大切なのはプログラムの構成と方向性を理論的に決め, それがブレないようにテストケースを活用しながら構築していくことだとわかった.

6 謝辞

終始熱心なご指導を頂いた情報科学演習 D の松本 真佑, 桝井 晃基担当教員, また TA の方々には心より感謝の意を表します.

参考文献

[1] quwahara, Qiita, 2 単純な構文解析を Java で実装する, https://qiita.com/quwahara/items/9bf468ff4286b28d2a24, 2017 年 07 月 18 日