情報科学演習Ｄ

第2回レポート課題

提出年月日：平成30年10月27日

電子メール：u839996e@ecs.osaka-u.ac.jp

学籍番号　：09B16084

氏名　　：山野広大

1　課題 2 の目的

Pascal 風言語で記述されたプログラムから切り出されたトークン列から、プログラムが構文的に正しいかどうか判定するプログラムの作成。

2　プログラムの動作原理

2.1　プログラムで利用したデータの説明

ここでは本プログラムで使用したデータについての説明を行う。

名前	型	用途
IDList	List<Integer>	トークンの ID を保存する
LineNumberList	List<Integer>	トークンの行数番号を保存する
index	int	現在読んでいるトークンが何番目かを保存する
fr	FileReader	入力ファイルを読み取るために使用する
br	BufferedReader	同上
line	String	読み込んだ行の文字列を保存する
tokenInfo	String []	読み込んだ行を'\t' で区切って保存する
temp	int	String 型を int 型に変形するのに使用する

2.2　プログラムの流れ

1. 入力用のファイルを作成する
2. 入力ファイルを読み込みモードでオープンする　※ 1
3. 入力ファイルを 1 行読み込み、その行のトークンの ID、行数を各リストに追加する
4. 指導書に書かれている各構文要素について構文的な誤りがないかチェックする　※ 2
5. 標準出力に"OK" と出力する

※ 1　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、終了する
※ 2　構文的な誤りを発見した場合は"Syntax error: line" という文字列と共に最初に発見した誤りの行数番号を標準エラーに出力する

3　プログラムの実装方針

この章では上記のプログラムの流れの 3 の入力ファイルを 1 行読み込み、その行のトークンの ID、行数を各リストに追加する部分と、4 の各構文要素について構文的な誤りがないかチェックする部分についての説明を行う。

3.1　流れ 3 の説明

まず BufferedReader クラスの readLine メソッドで読み込んだ入力ファイルの行を line に代入する。そして line を split 関数で\t ごとに分け、それを tokenInfo に代入する。tokenInfo の 2,3 番目の要素がそれぞれトークンの ID、行数番号なのでまずは tokenInfo[2] を Integer クラスの valueOf メソッドで int 型に変え、それを temp に代入する。その後 temp を IDList に追加する。次に tokenInfo[3] についても同様に int 型に変えた後 lineNumberList に追加する。

3.2　流れ 4 の説明

構文誤りのチェックについてはまず指導書記載の EBNF に対応するメソッドを作成し、そのメソッドの中で 1 つ 1 つのトークンについて判定するという方法をとった。詳しくは以下で述べる。

3.3　EBNF をどのように構文解析可能したか

構文要素の解析は指導書に定義されている EBNF 式の各要素に対応するメソッドを作成して行った。run メソッドからプログラムを解析するメソッドが呼ばれ、その中でブロックを解析するメソッドが呼ばれというようなアルゴリズムで動作する。このアルゴリズムを用いることで EBNF が構文解析可能となる。

3.4　EBNF の文をどのようにメソッド化したか

3.4.1　メソッドの概要

各メソッド内で EBNF 式の要素を先頭から判定していく。
その要素の種類として特定のトークン（例：”program”,”;”）、構文要素の 2 種類に分けられる。
要素が特定のトークンであった場合は期待されるトークンの ID を引数としてトークンの判定を行う consume メソッドを呼び出す。consume メソッドについてはのちに説明する。要素が構文要素であった場合はその構文要素を解析するメソッドを呼び出す。ただし、指導書の EBNF 式では構文要素であっても、その構文要素の句が特定のトークンのみであった場合は簡略化のために直接 consume メソッドを呼び出す。（例：副プログラム頭部の要素に「手続き名」があるが、手続き名の要素は「名前」というトークンのみなので副プログラム頭部の解析メソッドでは「手続き名」を解析するメソッドを呼び出すのではなく直接 consume メソッドで名前トークンであるかの判定を行う。）
また、このように次々と構文要素を呼び出していっても最終的にはトークンの判定を行う必要があり、トークンの判定を終えた時点で index の値を 1 増やし次のトークンを読む準備をする。

3.4.2　consume メソッドについて

consume メソッドは現在解析しているトークンが、期待されているトークンと一致するかの判定を行う。呼び出し元で期待するトークンの ID を引数にとり、その ID と、IDList の index 番目の要素が一致するかどうかで判定を行う。一致した場合は index の値を 1 増やす。一致しない場合はエラー文を出力するメソッドを呼び出す。

3.4.3　選択の判定

選択の判定は基本的に構文要素の 1 番目の判定で実装した。if 文は、まず”if” を読んだ段階で分岐させて、その 4 文字後に”else” が来るかどうかで解析を行う。基本文は 1 番目の判定で複合文、入出力文の判定を行い、2 番目の判定で代入文、手続き呼び出し文の判定を行った。

3.4.4　繰り返しの判定 (0 回以上の出現)

繰り返しの判定が必要な構文要素は繰り返し前の要素を解析するメソッドのみを持つ。繰り返しの可能性がある構文要素は次の要素を判定すれば繰り返しが有るか無いかがわかる。繰り返しがあると判定されたら再帰的にそのメソッドを呼び出す。

3.4.5　選択任意の判定 (0 回または 1 回の出現)

任意選択の可能性を持つ構文要素はまずその構文要素を解析するメソッドを呼び出し、その構文要素の先頭の要素が任意選択の要素と一致したらその構文要素の解析メソッドを呼び出し、一致しなければ何もせずに次のプログラムへ進む。

4　まとめ

課題 2 では Pascal 風言語で記述されたプログラムから切り出されたトークン列から、プログラムが構文的に正しいかどうか判定するプログラムを作成した。EBNF の各要素に対応するメソッドを作成するという方法をとった。メソッド内でメソッドを呼ぶことが多いが、どれだけメソッドを呼んでも最終的にはトークンをチェックする必要があるためこのような書き方が可能になった。

5　感想

構文的な誤りを見つけた時、”Syntax error: line”という文字列と共に最初に発見した誤りの行数番号を標準エラーに出力した後プログラムを終了する際に、初めは exit メソッドを使用していた。exit メソッドを使用すると期待するエラー文と実際に出力されたエラー文が一致していてもテストに通らなかったため、RuntimeException でエラー処理を行うことにした。
index の値を増やす所と増やさない所の判断がうまくできなくて読みたい文字と現在読んでいる文字が違うといったことがあったため、その部分でやや苦労した。
繰り返しの処理の部分は比較的うまく作れたと思う。
メソッドがメソッドを呼ぶなど、やや複雑になったが、初めの段階で作成したメソッドを終盤で再使用できたことなど効率よく行えた部分があったことはよかったと思う。

謝辞

本課題に取り掛かる直前に教員の方に実装方針を軽くご指導いただいたため、スムーズに課題を終わらせることができました。本当にありがとうございました。