課題 1 レポート

学生番号 09B17020 河越　淳

2019 年 10 月 18 日

1　課題の目的

課題 1 では第一引数で指定された pas ファイルを読み込み、それをトークン列に分割する字句解析を行う。また、分割したトークン列は第二引数で指定された ts ファイルに書き出し、正常に処理が終了した場合は標準出力 (System.out) に”OK”を出力して、入力ファイルが見つからない場合は標準エラー (System.err) に”File not found”と出力して終了するという処理を行う。

2　課題達成の方針と設計

課題達成のための方針としては、まずファイルを一行づつ読み込み、その一行の中で先頭から順に文字を見ていく、そして次に示す順に処理をファイル内の文字が全て読み込まれるまでおこなう。
(ただし上の処理が行われた場合はその先の処理は行われずに、その先の文字を 1 から順に処理していく)

1. 注釈 {} かどうかを判別する。
2. ’ 文字列’(記号ではない) かどうかを判別する。
3. 数字かどうかを判別する。
4. スペースがないかどうかを判別する。
   （a）/以外の記号かどうかを判別する。
   （b）/かどうかを判別する。
   （c）名前かどうかを判別する。
5. スペースがある場合はとばして、1 に戻る。

上記の順に判別をしていくことで字句解析を行う。

3　プログラムの実装方針

次にプログラムの実装について詳しく説明する。
そのために、まずトークン列としての出力の仕方について説明する。
出力は次のような関数を読み込み、それの返り値を書き込むことで行う。
この関数の引数は string(プログラム中に現れたトークン) とトークンと行番号を引数とし、これにより Lexer で定義したトークン名、トークンの ID、および行番号の順でトークンがファイルに出力される。

```
1 public static String getLineOutput(String string, Token token, int
      lineNumber)
2 {String s=string+"\t"+token.toString()+"\t"+token.ordinal()+"\t"+lineNumber
      +"\n";
3 String output = s;
4 return output;}
```

この 2 つ目の引数の Token とは enum クラスで定義したもので token.toString() でトークン名を token.ordinal() でトークン ID を呼び出すことができる。トークンクラスは次のようになっており、すべてのトークンが登録されている。
しかし、SDIVE の/だけは個別で判別を行う。

```
1 public enum Token {
2   SAND("and"),
3   SARRAY("array"),
4   SBEGIN("begin"),
5   SBOOLEAN("boolean"),
6   SCHAR("char"),
7   SDIVD("div"),
8       '''中略'''
9 }
```

次に 2 で記した判別について説明を行う。

3.1 1

判別はまず、最初に注釈かどうかを評価する。判別の仕方としては下記のように一行目で { を判別し、} が出てくるまで、そのかっこ内の文字を飛ばすことで字句解析の中に注釈を含めないようにする。

```
1 if(line.charAt(charNumber)=='{'){//{ か どう か 判 別
2   charNumber += 1;
3 while(charNumber<line.length() && line.charAt(charNumber)!='}'){
4   charNumber += 1;}
5   charNumber += 1;
6 }
```

3.2 2

次に文字列かどうかを判別する。これは最初に' があるかどうかで判別し、次の' が出てくるまでの文字を文字列として出力する。ファイルに出力する際に 3 の最初に説明した getLineOutput 関数と write 関数を用いる。

```
1 else if(line.charAt(charNumber)=='\''){//文 字 か どう か 判 別
2   output+="\'";
3   charNumber += 1;
4 while(charNumber<line.length() && line.charAt(charNumber)!='\''){
5   output+=line.charAt(charNumber);
6   charNumber += 1;}
7 if(charNumber<line.length() &&line.charAt(charNumber)=='\''){
8   output+="\'";}
9 writeLine += getLineOutput(output, Token.SSTRING, lineNumber);
10 bw.write(writeLine);}
```

3.3 3

その次は数字かどうかを判別する。これは isDigit 関数を用いて実装しており、数字であれば数字でなくなるまでを一塊りとして出力する。

```
1 else if(Character.isDigit(line.charAt(charNumber))){//数 字 か どう か 判 別
2 while(charNumber<line.length() && Character.isDigit(line.charAt(charNumber
      ))) {
3   charNumber += 1;}
4 writeLine += getLineOutput(output, Token.SCONSTANT, lineNumber);
5 bw.write(writeLine);}
```

3.4 4

その後は空白かどうかを判別する。空白であれば飛ばして次の文字を見て、3.1 1 に戻る。空白でなければ記号または名前であるとして処理を行う。
処理の仕方としては、まず下にある for 文で名前、数字、文字以外のトークン全てをチェックしていく。

```
1 for(Token Token : Arrays.copyOfRange(Token.values(), 0, Token.SDOT.ordinal
      ()+1)){//Token全部見る
```

3.4.1 1

次にトークンの判別を行う。判別方法としては enum クラスにある記号を順に見ていく方法を用いる。
下のプログラムの 2、3 行目で、評価しているトークンが<、>、:である時に、評価している文字の次の文字が=であれば、その文字は<=、>=、:=の文字列であるとしており、これにより間違って<=、>=、:=を<、>、:だと判別しないようにしている。
5 行目でトークンが行を超えている場合は通らないようにしている。
6 行目で特殊記号であることを判別している。
7 行目で綴り記号の後ろが空白であることを利用して綴り記号であることを判別している。これらを行うことで数字、文字列、識別子以外のトークンであることの判別を行う。

```
1 if(!(
2   (Token==Token.SLESS || Token==Token.SGREAT || Token==Token.SCOLON)
3    && line.charAt(charNumber+1)=='=')
4      &&
5   (charNumber+Token.getToken().length()>=line.length()
6    ||Token.ordinal()>Token.SWRITELN.ordinal()
7    ||!Character.isLetterOrDigit( line.charAt(charNumber+Token.getToken().
         length() ) ) )
8   )
9   )
```

3.4.2 2

そして/の場合だけ別で/であるかどうかを判別する処理を次のように行い、トークン ID は SDIVD となるようにする。

```
1 else if(line.indexOf("/", charNumber)==charNumber){
2   writeLine += getLineOutput("/", Token.SDIVD, lineNumber);}
```

3.4.3 3

また、すでに登録されているトークンでなかった場合は名前であるとして処理をする。

4　まとめ

今回の字句解析では注釈、符号なし整数、文字列、識別子特殊記号、名前の 5 つに大別をすることで解析を行うことにした。そのために、1 行ずつ行の先頭から 1 文字を注釈、符号なし整数、文字列、識別子特殊記号、名前の順で判別していった。また、<=、\>=、:=などで必要があるときだけ一つ先の文字も合わせてみるようにした。これにより、字句解析のプログラムが完成した。

5　感想

今回の演習では java を用いたが、今まで java を使ったことがなかったためどのようなライブラリがあるのかが分からなく少し苦労した。しかし、今までに swift を使ったことがあったため、それと同じような感じのライブラリがないかなどを web で検索し、実装にうつすことができた。特にトークンの判別の際に列挙型をうまく使えたのがとても良かった。