情報科学演習Ｄ　課題1レポート

担当教員：松本　真佑
提出者：市川　達大
学籍番号：09B17010
メールアドレス：u243952e@ecs.osaka-u.ac.jp

提出年月日：2020 年 10 月 15 日

1　課題の目的

演習 D を通して、Pascal 風言語で記述されたプログラムを，アセンブラ言語 CASL 2 で記述されたプログラムに変換するコンパイラを作成する。今回の課題 1 では、その手始めとして Pascal 風言語で記述されたプログラムからトークン列を切り出す字句解析 (Lexer) プログラムを作成する。

2　課題の達成の方針と設計

2.1　外部仕様

Pascal 風言語で記述されたプログラム (pas ファイル) を受け取り、字句解析の結果を ts ファイルへ出力する。

2.2　方針と設計

以上、課題を達成するために主に以下の方針で設計し実装する。課題達成の要件として主に以下 3 つの課題がある。

1. ファイルの入力、トークンへの分割
2. トークン判定
3. ファイルへ結果出力

2.2.1　ファイルの入力、トークンへの分割

まず、与えられた pas ファイルを読み込む必要があるので、入力をどのような方法で行うかを考える。行数カウントの必要性とトークンへの分割を考えると入力 pas ファイルを 1 行毎に読み込み、行毎にトークン分割を行うのがメモリも圧迫しないので最適である。また、トークン分割の方法として、はじめに思いつくのは空白文字ごとに分割してトークンとして切り出すことであるがこの場合トークンが連続して記述されている場合などでエラーが起るので、読み込んだ 1 行を先頭から 1 文字ずつ読み込んでいき各トークンが終了した時点でトークンとして切り出す、という方針で設計する。

2.2.2　トークン判定

トークン判定では、まず判定を行うために字句とトークン名の一覧データを配列として保持し、前処理で切り出したトークンがどのトークンと対応するかを判定する。またこの処理はトークンが切り出される毎に逐次実行することで切り出したトークンの保持をする必要をなくす。

2.2.3　ファイルへ結果出力

結果の出力は、トークンを分割し、逐次トークン判定を行ったところで字句、トークン名、ID、行数とともに出力ファイルへ上書きしていく。

3　プログラムの実装方法

上記の設計方針をもとに具体的にプログラムを実装していく。

3.1　ファイルの入力、トークンへの分割

まずファイルを 1 行ずつ読み込むのは BufferedReader クラスの readLine メソッドを用いて入力ファイルを 1 行ずつ読み込んでいく。またこの時、ファイルを開くのに失敗した時のエラー処理も行う。

```
1  try(BufferedReader in = new BufferedReader(new FileReader(new File(inputFileName)))){ //
       ファイル読み
       込み
2  .
3          while((line = in.readLine()) != null) {
4      .
5      .
6      }
7  .
8
9  }catch(IOException e) {
10                             System.err.println("File␣not␣found");
11 }
```

次にこの読み込んだ 1 行を先頭から順番に 1 文字ずつ読み込んでいくことでトークンへの分割を行う。この時以下の流れでトークンへの分割を行う。

1. トークンの 1 文字目を見て以下 5 種類に分類する。
   (a) 1 文字目が英字→名前 or 英字のトークン
   (b) 1 文字目が数字→符合なし整数
   (c) 1 文字目が{ →　注釈
   (d) 1 文字目が’ →　文字列
   (e) 1 文字目が空白記号　→ 読み飛ばす
   (f) 1 文字目がきごう　→　記号のトークン
2. 2 文字目以降を見てトークンの終了判定を行う

上記のルールに基づき、以下のような流れでトークンを判定していきトークン判定終了後、ファイルへ出力していくといった流れになる。

```
1  for(//分割した文字を 1文字ずつ見ていく) {
2
3
4          //1 文字目の場合
5          if(alpha_flag == 0 && num_flag == 0 && annotation_flag == 0 && symbol_flag == 0
               && string_flag ==0) {
6
7                  //1 文字目が英字 or 整数 or 注釈 or 文字列 or 空白記号 or 記号かを判定
8
9          }
10
11         //2 文字目以降を見る
12         if(char_num >= input_line_char.length −1) {//行の最後の文字かどうかを判定
13                         ..//次の行を取りに行く
14         }
```

```
15          else {//行の最後でない場合
16                              ..//トークン終了判定
17          }
18      .
19      .
20      //トークンがデフォルトで規定されているトークンであるかを判定
21      .
22      .
23      //出力へ
24  }
```

3.2 トークン判定

トークンへの分割を行った後、そのトークンがどの種類のトークンであるかを判定する必要がある。このトークン判定はトークンのデータとして配列へ格納したデータと参照し、該当するものがあるかどうかを判定する。該当するものがなければ名前として判定する。

```
1  //トークンデータ
2
3  final String[] token_source = {"and", "array", "begin", "boolean", "char", "div", "/", "do", "
       else", "end", "false", "if","integer", "mod", "not", "of", "or", "procedure","program"
       , "readln", "then","true", "var", "while","writeln", "=", "<>", "<", "<=", ">=", ">", "+
       ", "-", "*", "(", ")", "[", "]", ";", ":", "..",":=",",", "."};
4
5  final String[] token = {"SAND","SARRAY","SBEGIN","SBOOLEAN","SCHAR","SDIVD","SDIVD","SDO",
       "SELSE","SEND","SFALSE","SIF","SINTEGER","SMOD","SNOT","SOF","SOR","SPROCEDURE","
       SPROGRAM","SREADLN","STHEN","STRUE","SVAR","SWHILE","SWRITELN","SEQUAL","SNOTEQUAL
       ","SLESS","SLESSEQUAL","SGREATEQUAL","SGREAT","SPLUS","SMINUS","SSTAR","SLPAREN","
       SRPAREN","SLBRACKET","SRBRACKET","SSEMICOLON","SCOLON","SRANGE","SASSIGN","SCOMMA
       ","SDOT"};
```

3.3 ファイルへ結果出力

トークン判定が終了したら次はファイルへの出力を行う。今回はトークン判定が終了したものをメモリで保持したくなかったので判定終了ごとに出力ファイルへ書き出していく。

```
1   //ファイルへ書き出し
2  if(symbol == 1) {
3          pw.println(j_token + "\t" + token[token_number] + "\t" + token_id + "\t" +
               line_count );
4          symbol = 0;
5  }
6  else if(string_flag == 1) {
7          pw.println(j_token + "\t" + "SSTRING" + "\t" + "45" + "\t"+ line_count);
8          string_flag = 0;
9  }
10 else if(num_flag == 1) {
11         pw.println(j_token + "\t" + "SCONSTANT" + "\t" + "44" + "\t" + line_count );
12         num_flag = 0;
13 }
14 else {
15         pw.println(j_token + "\t" + "SIDENTIFIER" + "\t" + "43" +"\t" + line_count );
16 }
```

上記 3 つのプロセス（トークンへの分割、トークン判定、ファイルへ結果出力）を入力ファイルが終端文字を読み込むまで続けることで、入力ファイルの字句解析を完了することができる。

4　まとめ

今回の課題 1 では Pascal 風言語で書かれたソースコードを字句解析をするプログラムを作成することで、「ファイルデータの処理方法」、「コンピュータに言語を理解させる」という自然言語処理に通づるものを学習することができた。

5　感想

今回の課題では、比較的用意なコードであったのでクラス分けを行わなかったので、もう少しオブジェクト指向に基づいて役割ごとにクラス分けをするべきだと思った。また、クラス分けをしなかったせいでトークンの誤判定バグの原因を修正するのに時間がかかったので、やはりオブジェクト指向に基づいたコードを書くべきであった。またリファクタリングする時間があれば修正したい。