情報科学演習Ｄ　課題１レポート

氏名　山久保孝亮
所属　大阪大学基礎工学部情報科学科ソフトウェア科学コース
メールアドレス　u327468b@ecs.osaka-u.ac.jp
学籍番号　09B22084

提出日　2024 年 10 月 14 日
担当教員　桝井晃基　松本真佑

1　システムの仕様

課題 1 で作成したプログラムの仕様は以下の通りである.

- Pascal 風言語で記述された pas ファイルを入力として，字句解析の結果である ts ファイルを出力する．また，開発対象の run メソッドの第一引数は pas ファイル，第二引数は ts ファイルを指定する.
- 正常に処理が終了した場合は文字列”OK”を返し，入力ファイルが見つからない場合は文字列”file not found”を返す.

2　課題達成の方針と設計

課題 1 は字句解析と，解析したトークンをそれに対応する文字列とともに ts ファイルに書き込む処理に分けられる.

2.1　字句解析の処理

字句解析の処理は以下のオートマトンに基づいて開発した.

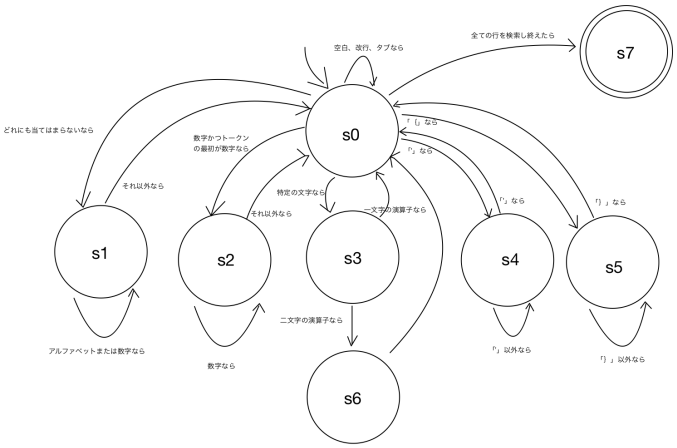

図 1: 字句解析部分のオートマトン

このオートマトンは合計 8 つの状態が存在し, 初期状態は s0, 受理状態は s7 である. それぞれの有効辺の条件分岐は入力された pas ファイルの一文字について判定している．状態が変わるとその次の文字について調べ，pas ファイルの最後の文字になると受理状態に遷移する．s1 から s0 に遷移するとき以外，s0 に遷移する際に ts ファイルに出力する処理を行う.
また，ここでは pas ファイルの一文字を取り出してそれについて条件分岐を作成しているが，トークンが複数の文字で構成される場合，読み込んだ文字を保存しておく必要がある．そのために token_loaded_letter というリストを用意し，トークンの終わりが確定するまで各文字を記憶しておく．そして確定してから各要素を連結して文字列に変換し，そのリストを初期化する．以降この処理を文字列連結の処理と呼ぶ.

2.2　ts ファイルに書き込む処理

ts ファイルに出力する処理を行うために，ソースコード上の字句，字句解析機上でのトークン名，ID のそれぞれを格納するリストを事前に用意しておく．具体的には，ソースコード中の字句を token，字句解析

機上でのトークン名を token_name，ID を token_id というリストを用意しておく．token を含め，これらのリストの要素の順番はすべて対応している．例えば，token,token_name,token_id の 0 番名にはそれぞれ and,SAND,0 が格納されているように，それぞれのリストのインデックスに対応する文字列が格納されている．これによりインデックスのみでトークン名と ID を取り出すことができる．また，div と \ はどちらも格納し，token_name でも同じトークン名で，token_id でも同じ ID で格納した．
そして，取り出したトークンが token の内の要素と一致するかを判定し，token_name と token_id に格納されている対応する文字列を書き込む．また，この処理は 2.1 のオートマトンの s1 以外の状態で処理されるので，同じ記述を繰り返すことを避けるために関数として定義した．

3　実装プログラム

2 で記述した方針で作成するために，ここでは pas ファイルから文字を取り出す処理，字句解析のオートマトンのそれぞれの状態の処理，ts ファイルへの書き込み処理の 3 つについて記述する．

3.1　pas ファイルから文字を取り出す方法

pas ファイルから各文字を一字ずつ評価するために，pas ファイルから一行ずつ取り出し，各行から一字ずつ取り出す処理を行う．以下はその部分のプログラムを抜粋したものである．

```
final List<String> buffer = Files.readAllLines(Paths.get(inputFileName));
String line = buffer.get(i) + "\n";
while(i<buffer.size()) {
    String line = buffer.get(i) + "\n";
    while(j < line.length()) {
        char c = line.charAt(j);
        j++;
    }
    i++;
}
```

これによって，c にプログラムの各文字が格納されていくことになる．また，2 行目で line の最後に \n を追加しているのは Files.readAllLines() では改行の文字が含まれないためである．

3.2　字句解析のオートマトンの処理

まず図 1 のオートマトンの各状態に遷移するための条件分岐について記述する．3.1 で取得した c について評価してどの状態に対応するかを判定する．以下はその部分のプログラムを抜粋したものである．

```
    //s0
    if(c == ' ' || c == '\t'|| c == '\n') {
    }else if(c == '{'){//s5
    }else if(c == '\'') {//s4
    }else if("/=<>+-*()[]:;.,".indexOf(c) !=-1){
        if((c == '<'||c == '>'|| c == ':'|| c == '.')&&( j + 1 < line.length()))
            {//s6
        }else {//s3
        }
    }else if("0123456789".indexOf(c)!=-1 && !loaded_token_letter.isEmpty() && !
        Character.isAlphabetic(loaded_token_letter.get(0).charAt(0))){//s2
```

```
10      }else {//s1
11      }
```

この部分は 3.1 の二つの while 文中に記述されているので，pas ファイルの文字数分だけ実行される．5 行目と 9 行目の”文字列”.indexOf(c) は文字列中に c が含まれているかどうかを判定しており，これで数字と特定の文字列が含まれているかを判定している．
5 行目の s3 に遷移するかどうかを判定する際の文字列では，指導書の表 6 のトークン一覧に記載されていた記号を羅列したものであり，図 1 のオートマトンの特定の文字というのはこの””で囲まれた文字列に対応する．また，6 行目の s6 に遷移するかどうかを判定する際の文字列では，トークン一覧に記載されていた二文字の記号の内の一文字目を羅列したものである．
9 行目の s2 に遷移するかを判定する条件文では，取り出した文字が数字であるかだけでなく，切り出し中のトークンの一文字目がアルファベットであるかを判定することで取り出した数字が符号なし整数の一部なのか，識別子の一部なのかを判定している．また，isEmpty() を使用しているのは loaded_token_letter に何も格納されていないにもかかわらず一文字目を調べてしまうことを避けるためである．

3.2.1 s1 の処理

ここでは，識別子のトークンを切り出す処理を行う．2.1 で記述したように token_loaded_letter にその文字 String 型に変更して追加する．この状態に遷移した時点ではトークンの終わりであるかがわからないのでここで書き込みは行わない．

3.2.2 s2 の処理

ここでは，符号なし整数を切り出す処理を行う．符号なし整数の終わりは数字ではない文字が来るまでなので，それまで do-while 文を用いて先読みする．この時の条件文は ”c が「0 から 9」に含まれない ” である．先読みしている間はそれぞれの文字を loaded_token_letter に追加して j をインクリメントしていく．繰り返しから抜けた際には j は符号なし整数の終わりの数字の次の文字を指しているので，j を一つ減らす．そして文字列連結の処理を行い，writeToken() を呼び出す．このとき，行番号を表す引数には i+1 を格納する．i ではなく i+1 を格納するのは，i は 0 から始まっているため行番号に変換するにはすべて 1 を足す必要があるためである．

3.2.3 s3 の処理

ここでは，一文字の演算子のトークンを切り出す処理を行う．まず最初に loaded_token_letter に格納されているかどうかを調べる．これは，演算子がある時点でそれ以前の文字列がトークンとして確定するためである．そして s2 と同様に書き込みの処理を行う．

3.2.4 s4 の処理

ここでは，文字列を切り出す処理を行う．文字列の終わりを表す「 ’ 」が現れるまで do-while 文を用いて先読みする．この時の条件文は ”c が「 ’ 」ではない ” である．先読みしている間はそれぞれの文字を loaded_token_letter に追加して j をインクリメントしていく．繰り返しから抜けた際には j は文字列の終わりを表す「 ’ 」を指しているので，j に対して変化は加えない．そして s2 と同様に書き込みの処理を行う．

3.2.5　s5 の処理

ここでは，注釈に対する処理を行う．s4 と同様，注釈の終わりを表す「}」が現れるまで do-while 文を用いて先読みする．この時の条件文は ”c が「}」ではない ” である．注釈の内容は送信しないので，繰り返しの中では j をインクリメントするだけである．繰り返しから抜けた際には j は注釈の終わりを表す「}」を指しているので，j に対して変化は加えない．そして s2 と同様に書き込みの処理を行う．

3.2.6　s6 の処理

ここでは，二文字の演算子になり得るトークンを切り出す処理を行う．具体的には，c とは別に next という変数に c の次の文字を読み込む．これにより，c と next を合わせて二文字の演算子であるかを判定することができる．二文字の演算子であればその二つの文字を連結して，一文字の演算子であればそのまま s2 と同様に書き込みの処理を行う．

3.3　ts ファイルに書き込む処理

ts ファイルに書き込むための処理として，writeToken() を定義した．引数は以下の通りである．

```
1 void writeToken(String token_str, List<String> token, List<String> token_name, List<
      String> token_id, int line_counter, BufferedWriter writer)
```

token_str はオートマトンによって確定したトークンの文字列である．また，line_counter は token_str が記述されていた行番号を表す．
具体的な処理内容としては，まず for 文を用いて token の要素数と同じだけ繰り返しを行う．繰り返しの中では token_str が token の i 番目の要素と一致しているかを判定する．一致していれば tokenName と tokenId という変数に，token[i] に対応する文字列を token_name,token_id から格納する．token_str が符号なし整数と文字列であればそれに対応する文字列を前述の二つの変数に格納する．また，このときの条件式はそれぞれ「token_str.matches(”\\d+”)」と「token_str.indexOf(”\”) !=-」であり，それぞれ正規表現での数字と,「 ’ 」が含まれているかどうかで判定している．そして write() を用いて ts ファイルに書き込む．

4　考察

課題 1 から私が考察した内容は，正規表現ではなく決定性有限オートマトンで実装した理由についてである．正規表現では，指定したパターンがマッチするまでバックトラックなどを用いるためマッチングの組み合わせパターンが膨大になってしまう可能性がある．[1] 一方，決定性有限オートマトンでは全ての状態に対して遷移が定義されているので，バックトラックなどを用いることなく処理を行える．以上の点より，私は今回の課題で決定性有限オートマトンを採用した．

5　感想

今回の課題を通して字句解析機の作成方法を学ぶことができた．符号なし整数と識別子の定義を間違えて認識して苦戦したので，最初にオートマトンの仕様を完全に決定できるとスムーズに開発が進むと感じた．

参考文献

[1] https://blog.shin1x1.com/entry/regex-performance 10/14 アクセス