情報科学演習Ｄ　課題2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 11 月 4 日
担当教員　桝井晃基　松本真佑

1　システムの仕様

課題 2 の外部使用は以下のとおりである．

- 第一引数で指定された ts ファイルを読み込む．
- 構文が正しい場合は文字列”OK”を返し，正しくない場合は”Syntax error: line”という文字列を返す．ただし，line の部分には最初に誤りがあった行番号を表し，複数の誤りがあったとしても最初の誤りのみ出力する．また，入力ファイルが見つからない場合は”File not found”を返す．

2　課題達成の方針と設計

今回の課題の方針としては，指導書に記述されている構文定義の左辺に対応するメソッドをそれぞれ作成し，再帰降下法により構文解析をした．それぞれのメソッドは全て int 型で定義し，返り値が 0 のときそのメソッド内の構文定義は正しいことを表し，返り値が 0 以外のとき構文の誤りがあった行番号を表す．具体的には，次のトークンとそのメソッド内で期待されるトークンを比較し一致していれば構文が正しく，一致していなければその時点での行番号を返す．
また，文の構文定義など，次の一つのトークンを読んだだけでは if-else の文なのか if だけの文なのか判断できないような場合は左くくり出しを行う．具体的には文の構文定義では，if から複合文までの定義が同じであるため，そこまでは同じ条件分岐として扱い，そこから再び次のトークンを読むことによる構文の判定を行う．また，行番号はどのメソッドでも使用できるようにするためグローバル変数として ts_line_number を定義している．また，この変数は 0 に初期化されている．

3　実装プログラム

run() メソッドでは ts ファイルの内容を列ごとにリスト buffer に格納し，今回の構文解析のプログラムが記述された parse() メソッドに引数として渡す．この parse() の返り値が 0 であればこのメソッドが正常に終了したことを表すので文字列”OK”を，返り値が 0 でなければその値がエラーが最初に発生した行なのでこれを外部仕様に記述した文字列とともに出力する．
parse() では，リスト buffer の ts_line_number 番目の要素を文字列 line に格納する．このとき，ts_line_number の値を 1 インクリメントする．課題 1 より，この文字列はトークン列とその情報を表すが，すべてタブ文字で区切られているため split() を使って配列 parts に格納する．今後，次のトークンを取り出すための処理は以上の操作と同じ内容を繰り返す．その後 program() を呼び出すが，この処理は以下の 3.2 の処理と同じであるため後述する．そして，ts_line_number と buffer の長さが同じになれば 0 を返す．
2 の方針に従ってプログラムを実装すると，今回のプログラムは以下のパターンのプログラムに分類できる．

1. 次に期待されているトークンが””で囲まれた予約語である場合．
2. 次に期待されているトークンが構文要素名である場合．
3. 中括弧 {...} で囲まれた 0 回以上の繰り返しの場合．
4. 角括弧 [...] で囲まれた 0 回または 1 回の出現の場合．
5. 次のトークンを読んだうえでどの句を選択するかを決定する必要がある場合．

以下でそれぞれの実装方法について記述する．

3.1 次のトークンが予約語である場合

このパターンのプログラムを一般化したものは以下のようになる．

```
String line = buffer.get(ts_line_number++);
String[] parts = line.split("\t");
if (!parts[2].equals("tokenID")) {
    return Integer.parseInt(parts[3]);
}
```

上述のとおり，ts ファイルから取り出されたトークンとその情報 (pas ファイル上でのトークン，字句解析器上でのトークン名，トークン ID，行番号) はそれぞれ parts の各要素に格納されている．今回の課題を通して，これらの内用いられるのはトークン ID と行番号の二つ，即ち parts[2] と parts[3] である．parts[2] 即ちトークン ID が想定されたトークン ID と一致しているかどうかを if 文を用いて判定している．ここでは一般化のため，equals() の引数の文字列は tokenID となっているが実際には次に期待されるトークンの ID となる．一致していなければこの時点でこのときの行番号 parts[3] が return される．そして一致していれば何も return されずに次の処理に進む．

3.2 次のトークンが構文要素名である場合

このパターンのプログラムを一般化したものは以下のようになる．

```
int result = syntax_definition(buffer);
if (result != 0) {
    return result;
}
```

ここで syntax_definition() は，それぞれの構文定義のメソッドを一般化したものである．int 型の変数 result に int 型で定義されたそれぞれのメソッドの返り値が格納されるので，result が 0 でない即ち呼び出したメソッド内で構文定義に誤りがあった場合にはその値が出力する行番号なので return する．0 であれば何も return されずに次の処理に進む．

3.3 0 回以上の繰り返しの場合

このパターンのプログラムを一般化したものは以下のようになる．

```
while(true) {
  line = buffer.get(ts_line_number++);
  parts = line.split("\t");
  if(!parts[2].equals("tokenID")) {
    ts_line_number--;
    break;
  }
//以下にさらに処理が続く

}
```

ここでは 3.1 のパターンから始まる構文定義を 0 回以上繰り返す場合を記述している．ここでは無限ループを用いて break されるまで繰り返し続けるという処理を採用した．さらにループの最初の予約語 (構文要素名) が，構文定義が間違っているという条件分岐に入った際にはその時の行番号を return せずに break 文で抜けられるようにした．ただし，このまま抜けるだけだと，直前に判定されたトークンを次の処理で判定することができないので ts_line_number を 1 デクリメントした．

3.4　0 回または 1 回の出現の場合

このパターンのプログラムを一般化したものは以下のようになる．

```
String line = buffer.get(ts_line_number++);
String[] parts = line.split("\t");
if(parts[2].equals("tokenID")){//1回繰り返す時の処理が続く

}else{//0回繰り返す時
  ts_line_number--;
}
```

ここでは 3.1 のパターンから始まる構文定義が 0 回または 1 回出現する場合の処理の一部を表す．次のトークンを先読みすることによって [...] で囲まれた部分の構文定義の一つ目と一致するか否かによって分岐を実現している．1 から 2 行目によって得られたトークンが想定されるトークンと一致するか否かによって条件分岐をしている．一致している場合は一回の出現であるとみなし，その後さらに [...] 内の構文解析が続く．一致しなかった場合は 0 回の出現とみなし，このとき読んだトークンは本来次の解析の際に読まれるはずであったトークンなので ts_line_number を 1 デクリメントした．

3.5　次のトークンを読んだうえでどの句を選択するかを決定する必要がある場合

このパターンのプログラムを一般化したものは以下のようになる．

```
String line = buffer.get(ts_line_number++);
String[] parts = line.split("\t");
if(parts[2].equals("tokenID1")){//tokenID1と一致する際の処理が続く

}else if(parts[2].equals("tokenID2")){//tokenID2と一致する際の処理が続く

}else{
  ts_line_number--;//どれにも一致しない場合の処理が続く

}
```

ここでは，3.1 から始まる構文定義によって解析する対象が異なる場合，即ち構文定義が”または”で区切られている場合の処理の一部を表す．3.4 と同様に，次のトークンを先読みすることによってそれぞれの分岐の場合の構文定義の一つ目と一致するか否かによって分岐を実現している．このプログラムの例はそれぞれの構文定義の一つ目のトークンが tokenID1,tokenID2, それ以外の 3 パターンに分類できるときの例である．1 から 2 行目で得られた次のトークンが tokenID1 と tokenID2 に一致する場合は，それぞれの条件分岐の中でその後の解析が進む．どれにも一致しない場合は，このとき読んだトークンは本来次の解析の際に読まれるはずであったトークンなので ts_line_number を 1 デクリメントした．

4　考察

今回の課題では LL 解析により構文を解析したが，LL 解析を実現するためには最も左の非終端記号に適用すべき生成規則を先読みした記号から一意に定める必要がる．具体的には，指導書の構文定義に左くくり出し等を使用して

5　感想

今回の課題を通して感じたこととしては，今後の課題ではテストファーストな開発を心がけていこうということである．私はウォータフォール型の開発をしてしまったため，ts_line_number の処理のデバッグにかなり時間がかかってしまった．slack で紹介されていたようにテストファーストな開発をすることで問題が発生している箇所が明確になるのでデバッグがかなり楽になると感じるので，今後の課題３と４では活用していきたいと思う．

参考文献

[1] 2024 年度情報科学演習 D 指導書

[2] 辻野嘉宏 (2019) コンパイラ第二版 オーム社