課題名　：　情報科学演習 D　課題 1 レポート

氏名　：　ブアマニー　タンピモン
学籍番号　：　09B20065
メール　：　u946641i@ecs.osaka-u.ac.jp

提出年月日　：　2022 年 10 月 20 日

1　システムの仕様

この課題には Pascal 風言語で記述されたプログラム (pas ファイル) からトークン列を切り出すプログラム (ts ファイル) を作成した。開発対象字句解析器のメソッドでは Lexer.run(String,String) で、一つ目の String は pas ファイルのディレクトリであり、二つ目の String は ts ファイルのディレクトリである。pas ファイルが見つからない場合は標準エラー (System.err) で”File not found”と出力し、ts ファイルが作らない。しかし、ts ファイルが見つければエラーがあってもすべてが ts ファイルに変更される。

2　課題達成の方針と設計

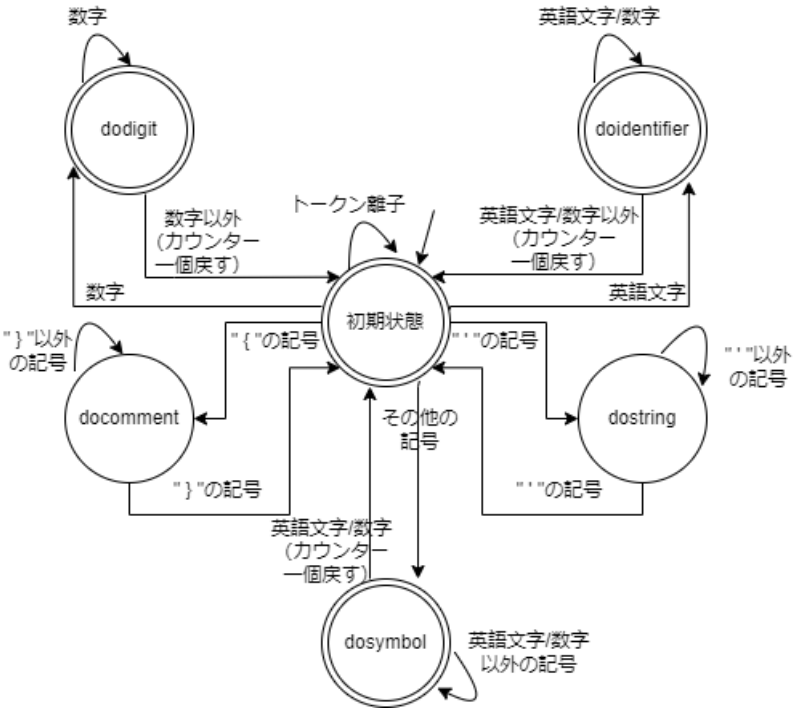

図 1: Lexer のオートマトン

開発対象字句解析器の概念とは pas ファイルから一つずつの行列を引き出し、図の Lexer のオートマトンで見れる通り、初期状態から始まり、行ごとにトークンの前頭から digit(記号なし整数),string(文字列),identifier&keyword(識別子と予約語),symbol(記号),comment(注釈) から種類を区別して、それぞれの方法でトークンを切り出す。そしてから、そのトークンの字句からトークン名と ID を検出して、行数とともに ts ファイルに追加し、次のトークンに進む。

2.1　状態の遷移（トークンの分け方）

初期状態からそれぞれのトークン状態に移動するには、トークンの特徴によって前頭からには５つの種類に見分けることができる。

a. digit(記号なし整数)

digit には一個ずつ入れると、0-9 の文字の連続である。0-9 文字を含まれるほかのトークンもあるが、digit のみは 0-9 文字から始まる。このように、トークンの始まりに文字 0-9 が表すと、digit であるとわかる。また、一つのトークンとして見るが、次の文字が 0-9 以外であればこれまでが digit トークンでありと ts ファイルに表示する。

b. string(文字列)

string の特徴は「’」から始まり、「’」で終わることである。このように「’」が見つかると文字列として扱い、「’」までがの文字列要素を string として保存する。最後の「’」が見つかると string トークンとして ts ファイルに表示するが、先頭の「’」で最後の「’」がないで行が終わるなら other トークンとして扱われる。

c. identifier&keyword(識別子と予約語)

まず identifier&keyword(識別子と予約語）には同じ先頭を持ち、a-z の文字である。しかし identifier(識別子) と keyword(予約語）の違いは identifier がユーザが定義された 0-9,a-z が含まれる自由な文字列であり、逆に予約語はプロぐラムが決めた「 ”program”, ”var”, ”array” , ”of”, ”procedure”, ”begin”, ”end”, ”if”, ”then”, ”else”, ”while”, ”do”, ”not”, ”or”, ”div”, ”mod”, ”and”, ”char”, ”integer”, ”boolean”, ”readln”, ”writeln”, ”true”, ”false”」24 個の文字列である。

d. symbol(記号)

トークン symbol の先頭は「 = < > + - * ( ) [ ] ; : . ,」という 14 個のシンボルである。しかし、記号トークンには一個だけでなく、「<> ¡= ¿= .. :=」2 個トークンも存在するため、よんで、すぐに一個トークンを判断できなく、すべての記号を読み取ってから前/次の記号の情報から判断するべき。

e. comment(注釈)

注釈の見分けは「{」より見分ける。「{」が来る場合は「{」が来るまで注意点としては注釈要素が行の終わりも含まれ、行が終わっても「}」が表すまでコメントされ続ける。

2.2 トークン離子

トークン離子には注釈、スペース、タブ、または行の終わり、また他のトークンが対応ではないシンブル「@,&」などが来るとプロぐラムは、トークンとして認めないため、プロぐラムが無視する。このように、オートマトンにはトークン離子が来る場合はそのまま初期状態に戻す。

3 実装プログラム

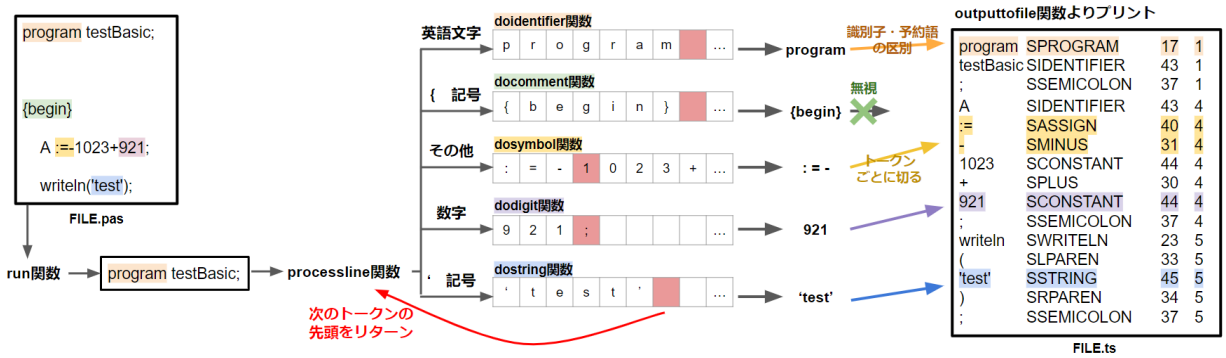

図 2: プロぐラムの仕組み

図で見れる通り、この課題には Lexer().run(inputFileName,outputFileName) により、プロぐラムが inputFileName から読み取って pas ファイルからトークン化し、ts ファイルに変更して outputFileName に出力する。run() 内に 7 つの関数に分かれて、それぞれの仕組みが以下に説明する。

3.1 run 関数

run 関数には outputFileName にファイルを作っておいて、inputFileName のファイルから 1 行ずつを processline 関数に入れて処理し、outputFileName の書き込むは processline で行う。processline の出力としてはファイルの読み込む状態であり、0 なら問題がないが、1 ならまだコメント内なので、次の行もコメント内になれるように先頭に {を入れる。しかし、出力が-1 なら string が終わらないまま行が終わって、エラーの意味な

ので即終了する。最後に、すべて処理終わって、全体に問題がなければ OK をコンソールに出力が、1 なら「}」がまだ終わってない、-1 なら「'」が終わってないのでエラーメッセージを出す。

3.2 processline 関数

processline 関数はそれぞれの line を入力されてもらって、currentletter により今の先頭部分がわかる。トークンの先頭を読み取ってそれぞれのトークンの種類に数字なら dodigit, 英語文字なら doidentifier,「'」なら dostring,「」なら docomment, その他なら dosymbol 関数に入れる。しかしスペースならトークン離子なのですぐつぎの文字に進む。その関数の出力は次のトークンの先頭であるため currentletter が最新する。

3.3 outputtofile 関数

この関数はトークンが特定できて outputFileName に出力するために使われる関数である。引数としては対象した出力ファイルの FileWriter とソースコード sourcecode, トークンの名 tokenname,id, とコードの行数 linenumber で、出力ファイルの次の行に書く。

3.4 dodigit 関数

この関数は先頭が文字列と文字先頭トークンの文字数を引数としてもらう。それで連続にある数字を最後まで探し、一つのトークンにする。やり方としてはトークンの先頭から次々の文字は数字かどうか確認し、数字なら次の文字を見るが、数字ではないなら先頭の文字から今の位置の一個前に数字列として outputtofile で出力し、次のトークンの文字位置がリターンする。

3.5 doidentifier 関数

トークンの先頭が英語文字なら識別子または予約語である。まず、識別子は予約語は２４つであり、それを token_identifier の配列に入れる。次にはトークンの最後の位置を探し、次の文字が英語文字または数字なら条件内である。条件外になるとその位置は次のトークンの先頭であり、今のトークンは予約語かを token_identifier にあるソースコード sourcecode と確認し、同じなら予約語として ts ファイルを出力し、全部違うなら識別子として outputtofile 関数を使って ts ファイルで表示する。それで次のトークンの位置をリターンする。

3.6 dostring 関数

文字列の場合は「'」から始まり、行内に「'」が来るまでその間がすべて文字列要素である。このように先頭から最後の「'」を探し続けて、見つかったら先頭と最後の「'」とともにトークンとして outputtofile 関数を使って ts ファイルに出力する。それで次のトークンの場所をリターンする。しかし、行が終わっても最後の「'」がない場合はコードエラーであり、エラーがあることを表示するために-1 をリターンする。

3.7 docomment 関数

コメントの先頭は「{」からはじめて「}」が来るまで無視続けるが、コメントの特徴としては次の行に行っても「}」が来るまでコメント状態のままである。このように、先頭の文字に「}」が見つけるまで調べ続けて、「}」がくるとそれは次のトークンの先頭である。しかし、行が終わっても「}」がまだ来ない場合は-1 を出力して次の行も「}」が来るまでコメントであることを表示する。

3.8 dosymbol 関数

トークンは英語文字ではないし、数字でもない、トークン離子でもない場合は記号の可能性がある。記号は基本 14 個とそれの組み合わせがあるが、トークン間にトークン離子がない可能性もあるので、記号の連続を全て切り取って、それでトークンずつに切る。まず、記号ではない文字を探し繰り返す。見つかったら記号の連続列ができる。それで一つの記号トークンは 2 文字または 1 文字であるため、例え先頭が「＜」なら 1 個の「＜」か 2 個の「＜＝」でもあるため、2 個から比較して見つからなかったら 1 個文字と比較する。1 個文字でも 2 個文字でも見つからない場合は無視して次の文字を見る。トークンを見つかる場合は outputtofile 関数で ts ファイルを出力する。最後に記号でない次のトークンの文字位置がリターンする。

4 考察や工夫点

4.1 トークンデータ管理

この課題にはは複数トークンの種類に区別しないといけなくて、その中にトークンデータの保存方法を扱いやすいようにどうすればいいかを考えないといけない。私が選んだやり方はオブジェクト Token を作って、その中にトークンに見分けるコード、トークン名、ID という 3 つの必要な情報を保存する。これはなぜなら一つの種類のトークンは同じ場所に揃ってるとコードがもっと理解しやすいし、トークン追加や、削除は簡単になる。それで Token が特徴により、token_identifier,singlesymbol,doublesymbol という 3 つの arraylist に保存されて、確認するときは先頭や順番により一つの種類のみ検索できる。

4.2 テストケース追加

このプログラムがもっと正しく動かせるように基本テストケースを載ってない次の 4 つのテストケースを追加して、どう解決するかを以下のように確認した。図で見れる通り、一つ目のテストケースは注釈のことであり、行が終わっても} がなければコメント状態をなりつつ、} が来るまで何も ts ファイルに追加しない。2 つ目のテストケースは登録されない記号が入る場合はプログラムエラーが出さずにプログラムが進んだがその記号がトークン離子として考える。3 つ目は {があって、} がない場合は Console に「OK」でなく「Program Error: No ending }」を表示する。4 つ目も 3 つ目と同じく、閉じる' がない場合は「Program Error: No ending ' 」を表示する。

pas ファイル	program testBasic;
	{begin
	writeln('test');}
	end.
		program testB}asic @;	program testBasic;
			{begin
			writeln('test');
			end.
				program testBasic;
				begin
				writeln('test);
				end.

実行結果	Location: .ts file program SPROGRAM 17 1 end SEND 8 4 . SDOT 42 4	Location: .ts file program SPROGRAM 17 1 testB SIDENTIFIER 43 1 asic SIDENTIFIER 43 1 ; SSEMICOLON 37 1	Location: Console Program Error: No ending }	Location: Console Program Error: No ending '

図 3: 追加テストケースの結果

5 感想

この課題にはレポートを書く途中でたくさんコードを訂正した。実行して成功からレポートを書く時に「{」,「"」 が一個しかないや「{」が複数行に使われるなどエラーができそうなケースを見つけた。プログラムがテストケースに成功してもできるだけエラーが全くないように訂正するべきと思う。これからもコードを書く前にちゃんとすべてのケースにエラーが出ないように注意する。