課題名　：　情報科学演習 D　課題 2 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 11 月 18 日

1　システムの仕様

この課題はコンパイラの二つ目のフェーズ、構文解析器（Parser）を作成する。概念で見ると一つ目のフェーズの字句解析系から出力したトークンを切り出す ts ファイルを入力として受け取り、構文の判定結果を出力する。構文解析器のメソッドは「Parser.run(String)」で String はファイル名となる。ファイル名から入力ファイルをみつけて確認し、構文が正しいなら「OK」を出力し、正しくないなら最初の誤りを見つけた行 X を「Syntax error: line X」で出力する。但し、ファイルが存在しない場合は「File not found」を標準エラーに出力する。

2　課題達成の方針と設計

課題 1 により、Pascal 風言語のプログラムがトークン化され、1 トークン 1 行 4 要素「＜入力トークン＞　＜トークン名＞　＜ ID ＞　＜行番号＞」から構成された ts ファイルを作成した。そのプログラムが構文的に正しいかどうか判定できるように、EBNF 形式の文脈自由文法で確認し、下降型解析の LL 解析でトークンが左から確認し、最初のエラーの行列を表示する。アイデアとしてはそれぞれの字句に対する関数を作成し、下降するたびに文脈自由文法通りに式にある構文要素の左順から最後まで確認する。エラーが発生する場合は次の関数に続かなく行数を戻り値として出力し、前の関数が自分の式にある関数からエラー状態を受け取って最後にプログラムを判定することができる。

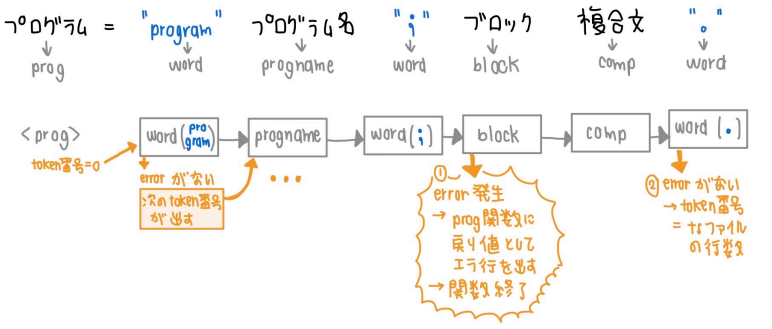

図 1: システムの流れ

LL(1) 文法とは 1 個のトークン（直後に来るトークン）の先読み込で構文解析可能な文法である。そのため、同一の左辺記号に対して、右辺の先頭トークンがすべて異なるために、「文」や「基本文」記号の式など 1 個で解析不可能の式を括り出して、問題を解決する。図 2 で見れる通り、「基本文」が「代入文、手続き呼び出し文、入出力文、複合文」という 4 つの選択可能な句で構文されて、「入出力文」の First 集合（先頭終端記号集合）は { ”begin” } 、{ ”readln”,”writeln” } で他の句と異なるが、「代入文」、「手続き呼び出し文」は二つとも識別子が先頭であるために、識別子読み取る場合は「識別子」として処理して、それで代入文の識別子抜きの句 replace の First 集合が { ”:=”}, 手続き呼出し文の識別子抜きの句 procedurecall の First 集合が { ”(” , ”;” } になるため、LL(1) が対応可能になる。

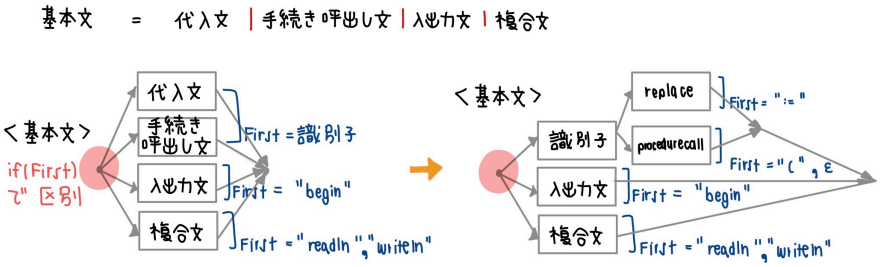

図 2: 括り出すにより LL(1) に変更する例

3　実装プログラム

このプログラムの考え方としてはそれぞれの字句が関数であり、現在のトークン番号を入力されて、そのトークンの字句が合ってるかどうか確認する。出力するのは Result というオブジェクトで、エラーが出す行数 errline と次確認するべきのトークン番号が出す。そのため、errline が””ではなく数字列の場合はエラーが発生する。しかし、エラーがない場合は次のトークンを確認する。

例を挙げると、プログラムの開始記号は Program で、それが以下の構文要素名と引用符 (””) で囲んだ文字記号からできた。

プログラム = ”program” プログラム名　”;” ブロック 複合文　”.” .

そのために、初めて確認されるのはトークン 0 で、関数 program に入れる。program 関数にには順番に”program” , プログラム名,　”;” , ブロック, 複合文,　”.”という五つの字句を確認するためにそれぞれの句の関数に呼び出し、出力が Result オブジェクトの answer 変数である。answer.errline が ”” のままの場合は answer.currenttoken という直後に来るトークンを次の句の関数に渡す。しかし、エラーが見つけるとすぐに Result オブジェクトの形でエラー番号を戻り値としてリターンする。”.”の関数まで確認してエラーがない場合は最後の字句”.”の戻り値を出力する。それぞれの句に対応する関数が以下のようである。

関数名	対応する字句	関数名	対応する字句
prog	プログラムの繰り返し	word	引用符 (””) で囲んだ文字記号
progname	プログラム名	block	ブロック
vardec	変数宣言	vardecl	変数宣言の並び
vardecl2	変数宣言の並びの繰り返し部分	varl	変数名の並び
varl2	変数名の並びの繰り返し部分	varname	変数名
type	型	ntype	標準型
atype	配列型	minnum	添え字の最小値
maxnum	添え字の最大値	num	整数
sym	符号	subdecg	副プロぐラム宣言群
subdec	副プログラム宣言	subhead	副プログラム頭部
procname	手続き名	tpara	仮パラメータ
tparal	仮パラメータの並び	tparal2	仮パラメータの並びの繰り返し部分
tparanamel	仮パラメータ名の並び	tparanamel2	仮パラメータ名の並びの繰り返し部分
tparaname	仮パラメータ名	comp	複合文
statementg	文の並び	statementg2	文の並びの繰り返し部分
statement	文	ifst	if 文
elsest	else 文	whilest	while 文
fundast	基本文	replace	代入文
variablewindex	純変数と添字付き変数	variable	変数
index	添字	procedurecall	手続き呼び出し文
equationl	式の並び	equationl2	式の並びの繰り返し部分
equation	式	pureequation	単純式
pureequation2	単純式の繰り返し部分	term	項
term2	項の繰り返し部分	factor	因子
relaop	関係演算子	addop	加法演算子
multop	乗法演算子	inout	入出力文
variablel	変数の並び	variablel2	変数の並びの繰り返し部分

表 1: 関数とその関数の対応する字句

また、それぞれの式は記号列が並ぶだけでなく、EBNF で、選択可能や、省略、繰り返し可能な部分がある。それに対するプログラムの対応は以下のように表す。

3.1 「 | 」選択可能な句

図 3 で見れる通り、選択可能な場合にトークンに対して複数な非終端記号が選択可能ため、それぞれの記号の First 集合（先頭終端記号集合）を求め、直後に来るトークンと同じ場合はその記号に選択する。しかし、どの記号でも First 集合にあるトークンが入力記号列と合ってない場合はトークンの情報から 4 列の行目のエラーが出る。

3.2 「 [ ] 」省略可能な句

図 4 で見れる通り、省略可能な句は、中にある式が記号列にあっても、なくてもいいので、[] の中にある式の First 集合を求めて、あるとしたら省略しないとして認める。しかし、First 集合と合ってない場合はエラーでなく、省略される可能性があるため同じトークン番号をリターンに出力して、次の式に確認する。

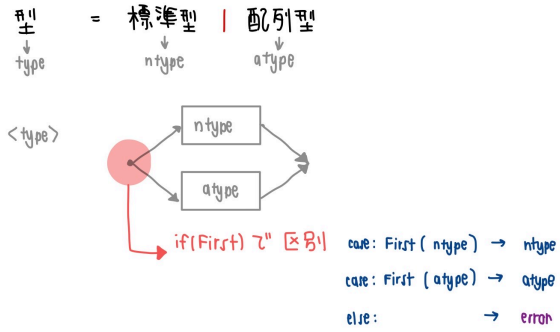

図 3: 「 | 」選択可能な句に対応

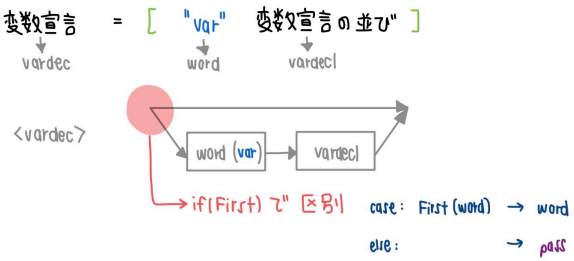

図 4: 「 [ ] 」省略可能な句に対応

3.3 「 { } 」省略可能な繰り返し句

図 5 で見れる通り、{ } の信号がある場合は省略可能な繰り返し句で、それの意味は、中にある式は省略しても、1 回でも、複数回でも可能である。そのため、実行し方法としては { } 内の式を新しい構文要素（名前は左辺に 2 をつける。例：tparal から tparal2 に）を作成する。この元の式は「[]」と同じく、中の式の First 集合とトークンの記号列と比べて、異なる場合は省略としてみとめて、前に既読できて、直後来るトークン番号をつぎの式にリターンする。しかし、同じ場合は新しい構文要素に呼び出す。新しい構文要素にはまず、{ } 内の式を一回既読し、それでまた「[]」と同じく、First 集合をを確認しまた同じトークンの場合は新しい要素を繰り返す。しかし、ない場合は次のトークンをリターンする。

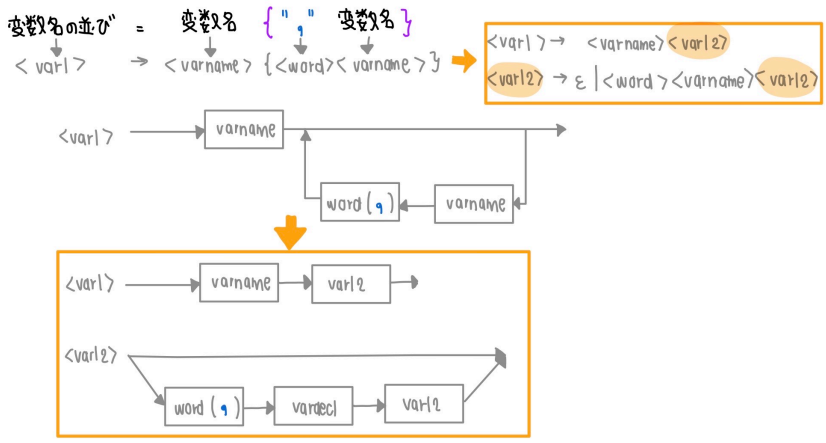

図 5: 「 { } 」省略可能な繰り返し句に対応

3.4 終了記号の句

最後に、終了記号の句の行動を説明する。word 関数はトークン番号と共に英語の文字列やシンブル (”¡=”,”:=”など) の終了記号を入力されて、直後のトークンはそれと合ってるなら今のトークンを既読として currenttoken+1(次のトークンを見る) を出力する。しかし、合ってない場合はエラーを出す。num 関数は sym 関数で記号を確認した後、記号なし整数を確認する。「記号なし整数」には文字ごとに確認する必要がなくて、文字列にある２つ目の字句解析器上でのトークン名で SCONSTANT で確認できる。これも同じく、確認が出来たら currenttoken(sym 関数から出たトークン番号)+1 をリターンする。しかし、SCONSTANT ではない場合はエラーを出す。

4 考察や工夫点

4.1 Result オブジェクトの作成

それぞれのプログラムの出力で、エラー/成功状態がわかるようにオブジェクト Result を作成する。オブジェクト Result の構造は以下の通りである。

Result オブジェクト

String errline= ””	:	エラーが出した行
int currenttoken	:	直後に来るトークンのトークン番号

関数から戻り値に、errline は””のままの場合はエラーがないが他の文字列ならエラーが発生する。currenttoken は直後に来るトークンのトークン番号なので、一つの関数に入ってでると、次の関数が currenttoken から確認する。そのため、その関数でも同じく、規律正しく実施することができる。

4.2 word 関数

引用符 (””) で囲んだ文字番号には、句の一種であり、”program”,”var”など色々式に構成させる。そのため、他の句と同じ形式で関数を持ち、Result オブジェクトがリターンできるため word 関数を作成し、トークン番号と確認したい文字列を入力ことで、次のトークン、エラーならエラー行をリターンする。

4.3 関数名

この課題には５４個の関数があって、再帰する部分が多いためできるだけ理解しやすいように関数名を決めた。そのため、基本の関数は構文要素の英語名で略して書く。また「.. の並び」なら後ろに「l」を付けて、「仮..」なら頭に [t] を付ける。また、中かっこ { } 繰り返しの部分の関数は基本の関数と同じ名前で、後ろに「2」をつける。そのため、関数を再帰で呼び出すときには整理になる。

5 感想

この課題で約７００行のコード、５４関数でできたため、コードの整理の必要性を学ぶことができた。プログラムにリターンでトークン番号を返すが、発生条件が整ってないや間違った変数をリターンすることで色々なデバッグしなければならないところがあった。変数の名前もルールを決めて書くと読むときが理解しやすくなって本当に嬉しかった。次回の課題でももっとキレイ、理解しやすいコードを書けるように頑張りたいと思う。