言語処理プログラミング

情報工学 3 年　20122041　中田継太

2022 年 11 月 28 日

目次

1　演習の目的

- コンパイラの基本的な構造とテキスト処理の手法を理解すること.
- 比較的大きなプログラムを作成する経験を得ること.

2　演習内容

MPPL で書かれたプログラムを読み込み,LL(1) 構文解析法により構文解析を行い，構文エラーがなければ入力されたプログラムをプリティプリントした結果を出力する。そして、構文エラーがあればそのエラーの情報 (エラーの箇所，内容等) を少なくとも一つ出力するプログラムを作成

3　プログラムの設計情報

3.1　全体構成

ここではどのようなモジュールがあるか，それらの依存関係について述べる。
プログラムは以下の 4 つのファイルで構成されている

■pprinter-list.c
main 関数があり、それぞれのモジュールを呼び出し、ファイルの読み込みから画面への出力を行うモジュール。scan-list.c を呼び出しファイルの読み込みを行う。pprinter-list.h を呼び出し、配列サイズや構造体 key といった定数、宣言を収得する。

■ebnf-list.c
EBNF 記法に基づいて、各規則に対しての構文解析処理関数があるモジュール。scan-list.c の scan() を呼び出し、token を解析する。pprinter-list.c に文法的誤りがあれば、エラーを返す。

■scan-list.c
ファイルの読み込みの初期化、トークンの収得およびファイルのクローズといった一連の処理を行うモジュール。scan() 関数は pprinter-list.c から呼び出され実行される。

■pprinter-list.h
定数トークンや関数の宣言を行うモジュール。定数トークンは scan-list.c や ebnf-list.c で呼び出される。

3.2　各モジュールごとの構成

ここでは使用されているデータ構造の説明と各変数の意味を述べる。

■key:**連想配列**
連想配列を用いた (言語によって辞書型、マップ型、ハッシュ テーブルと呼ばれることがある)。連想配列とはデータの場所を表す「キー」と、データの「バリュー」を対応付けて格納したデータ構造である。[1] 今

回は Strcut KEY を用いて、連想配列を実現させた。keyword はトークン KEYWORD の文字列を保持し、keytoken はトークン KEYWORD の TOKEN 番号を keyword に連結して保持している。

ソースコード 1　構造体 KEY in token-list.h

```
1  struct KEY {
2      char * keyword;
3      int keytoken;
4  } key[KEYWORDSIZE];
```

■変数

int numtoken[NUMOFTOKEN+1]　トークンの数を保持する変数

char *tokenstr[NUMOFTOKEN+1]　トークンの文字列の配列

int token　scan 関数で収得した TOKEN を保持する変数

ソースコード 2　各変数 in pprinter-list.c

```
1  int numtoken[NUMOFTOKEN+1];
2  char *tokenstr[NUMOFTOKEN+1];
3  int token;
```

int cubf　1 文字分の文字バッファ

int num_line　scan() で返されたトークンの行番号を保持する変数

int num_indent　プリティプリンタしたときにできたインデントの数を保持する変数

int num_attr　scan() の戻り値が「符号なし整数」のとき、その値を格納する

int num_then　then で何段落字下げを行った回数を保持する変数

enum State　直前の token の状態ををを示す列挙型変数。

char string_attr[MAXSTRSIZE]　scan() の戻り値が「名前」または「文字列」のとき，その実際の文字列を格納する

FILE *fp = NULL　開きたいファイルのファイル変数

ソースコード 3　各変数 in scan-list.c

```
1  int cbuf, num_line= 1;
2  int num_indent = 1;
3  int num_attr;
4  int num_then = 0;
5
6  enum PreState{
7      OTEHER,
8      SEMI,
9      THEN
10 };
11 enum PreState prestate = OTEHER;
12 char string_attr[MAXSTRSIZE];
13 FILE *fp = NULL;
```

3.3　各関数の外部仕様

ここではその関数の機能．引数や戻り値の意味と参照する大域変数，変更する大域変数などを記述する。

3.4　各関数の外部仕様 (今課題で新しく作成したもの)

3.4.1　print_indent 関数

■機能　プリティプリンタの際に改行を行いインデントを表示する。

■引数　なし。

■戻り値　なし。

ソースコード 4　print_indent()

```
1 void print_indent(void);
```

3.4.2　print_tab 関数

■機能　プリティプリンタの際にインデントを表示する。print_indent とことなり、改行はしない。

■引数　なし。

■戻り値　なし。

ソースコード 5　print_tab()

```
1 void print_tab(void);
```

3.4.3　init_string_atr 関数

■機能　変数 string_atr の中身をヌル文字に書きかえ初期化を行う。

■引数　変数 string_atr に格納された文字数。

■戻り値　なし。

ソースコード 6　init_string_atr()

```
1 void init_string_atr(int count);
```

3.4.4　set_token(int t) 関数

■機能　token をセットする関数

■引数　scan() で読み込んだ token。

■戻り値　なし。

ソースコード 7　set_token()

```
1 void set_token(int t);
```

3.4.5　parse_program() 関数

■機能　program の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 8　parse_program()

```
1  int parse_program();
2      // program ::= "program" NAME ";" block "."
```

3.4.6　parse_block() 関数

■機能　block の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 9　parse_block()

```
1  int parse_block();
2      // block ::= {variable_declaration | procedure_declaration} compound_statement
```

3.4.7　parse_variable_declaration() 関数

■機能　variable_declaration の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 10　parse_variable_declaration()

```
1  int parse_variable_declaration();
2      // variable_declaration ::= "var" variable_names ":" type ";" {variable_names ":"
           type ";"}
```

3.4.8　parse_variable_names() 関数

■機能　variable_names の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 11　parse_variable_names()

```
1  int parse_variable_names();
2      // variable_names ::= NAME {"," NAME}
```

3.4.9　parse_type() 関数

■機能　type の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 12　parse_type()

```
1  int parse_type();
2      // type ::= standard_type | array_type
```

3.4.10　parse_standard_type() 関数

■機能　standard_type の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 13　parse_standard_type()

```
1  int parse_standard_type();
2      // standard_type ::= "integer" | "boolean" | "char"
```

3.4.11　parse_array_type() 関数

■機能　array_type の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 14　parse_array_type()

```
1  int parse_array_type();
2      // array_type ::= "array" "[" NUMBER "]" "of" standard_type
```

3.4.12　parse_subprogram_declaration() 関数

■機能　subprogram_declaration の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 15　parse_subprogram_declaration()

```
1  int parse_subprogram_declaration();
2      // subprogram_declaration ::= "procedure" NAME [formal_parameters] ";" [
           variable_declaration] compound_statement ";"
```

3.4.13　parse_formal_parameters() 関数

■機能　formal_parameters の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 16　parse_formal_parameters()

```
1  int parse_formal_parameters();
2      // formal_parameters ::= "(" variable_names ":" type {";" variable_names ":" type}
           ")"
```

3.4.14　parse_compound_statement() 関数

■機能　compound_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 17　parse_compound_statement()

```
1  int parse_compound_statement();
2      // compound_statement ::= "begin" statement {";" statement} "end"
```

3.4.15　parse_statement() 関数

■機能　statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 18　parse_statement()

```
1  int parse_statement();
2      /*
3          statement ::= assignment_statement | condition_statement | iteration_statement |
4                        exit_statement | call_statement | return_statement | input_statement|
5                        output_statement | compound_statement | empty_statement
6      */
```

3.4.16　parse_condition_statement() 関数

■機能　condition_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 19　parse_condition_statement()

```
1  int parse_condition_statement();
2      // condition_statement ::= "if" expression "then" statement ["else" statement]
```

3.4.17　parse_iteration_statement() 関数

■機能　iteration_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 20　parse_iteration_statement()

```
1  int parse_iteration_statement();
2      // iteration_statement ::= "while" expression "do" statement
```

3.4.18　parse_exit_statement() 関数

■機能　exit_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 21　parse_exit_statement()

```
1  int parse_exit_statement();
2      // exit_statement ::= "break"
```

3.4.19　parse_call_statement() 関数

■機能　call_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 22　parse_call_statement()

```
1  int parse_call_statement();
2      // call_statement ::= "call" NAME ["(" expressions ")"]
```

3.4.20　parse_expressions() 関数

■機能　expressions の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 23　parse_expressions()

```
1 int parse_expressions();
2     // expressions ::= expression {"," expression}
```

3.4.21　parse_return_statement() 関数

■機能　return_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 24　parse_return_statement()

```
1 int parse_return_statement();
2     // return_statement ::= "return"
```

3.4.22　parse_assignment_statement() 関数

■機能　assignment_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 25　parse_assignment_statement()

```
1 int parse_return_statement();
2     // return_statement ::= "return"
```

3.4.23　parse_variable() 関数

■機能　variable の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 26　parse_variable()

```
1 int parse_variable();
2     // variable = NAME ["[" expression "]"]
```

3.4.24　parse_expression() 関数

■機能　expression の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 27　parse_expression()

```
1 int parse_expression();
2     // expression ::= simple_expression {relational_operator simple_expression}
```

3.4.25　parse_simple_expression() 関数

■機能　simple_expression の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 28　parse_simple_expression()

```
1 int parse_simple_expression();
2     // simple_expression ::= ["+"|"-"] term {adding_operator term}
```

3.4.26　parse_term() 関数

■機能　term の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 29　parse_term()

```
1 int parse_term();
2     // term ::= factor {multiplying_operator factor}
```

3.4.27　parse_factor() 関数

■機能　factor の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 30　parse_factor()

```
1 int parse_factor();
2     // factor ::= variable | constant | "(" expression ")" | "not" factor | standard_type
          "(" expression ")"
```

3.4.28　parse_constant() 関数

■機能　constant の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 31　parse_constant()

```
1 int parse_constant();
2     // constant ::= "NUMBER" | "false" | "true" | "STRING"
```

3.4.29　parse_multiplicative_operator() 関数

■機能　multiplicative_operator の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 32　parse_multiplicative_operator()

```
1 int parse_multiplicative_operator();
2     // multiplicative_operator ::= "*" | "div" | "and"
```

3.4.30　parse_input_statement() 関数

■機能　input_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 33　parse_input_statement()

```
1 int parse_input_statement();
2     // input_statement ::= ("read" | "readln") ["(" variable {"," variable} ")"]
```

3.4.31　parse_output_statement() 関数

■機能　output_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 34　parse_output_statement()

```
1 int parse_output_statement();
2     // output_statement ::= ("write" | "writeln") ["(" output_format {"," output_format}
          ")"]
```

3.4.32　parse_output_format() 関数

■機能　output_format の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 35　parse_output_format()

```
1  int parse_output_format();
2      // output_format ::= expression [":" "NUMBER"] | "STRING"
```

3.4.33　parse_empty_statement() 関数

■機能　empty_statement の文法を確認する関数

■引数　なし。

■戻り値　文法的誤りがあれば、1 を返す。なければ 0 を返す。

ソースコード 36　parse_empty_statement()

```
1  int parse_empty_statement();
2      // empty_statement ::= ε
```

3.4.34　programPrint() 関数

■機能　token が program であったとき、前の token によって表示位置を調整する。

■引数　str_atr に格納された文字数

■戻り値　TPROGRAM

ソースコード 37　programPrint

```
1  int programPrint(int count);
```

3.4.35　beginPrint() 関数

■機能　token が begin であったとき、前の token によって表示位置を調整する。

■引数　str_atr に格納された文字数

■戻り値　TBEGIN

ソースコード 38　beginPrint

```
1  int beginPrint(int count);
```

3.4.36　ifPrint() 関数

■機能　token が if であったとき、前の token によって表示位置を調整する。

■引数　str_atr に格納された文字数

■戻り値　TIF

ソースコード 39　ifPrint

```
1 int ifPrint(int count);
```

3.4.37　elsePrint() 関数

■機能　token が else であったとき、前の token によって表示位置を調整する。

■引数　str_atr に格納された文字数

■戻り値　TELSE

ソースコード 40　elsePrint

```
1 int elsePrint(int count);
```

3.4.38　thenPrint() 関数

■機能　token が then であったとき、前の token によって表示位置を調整する。

■引数　str_atr に格納された文字数

■戻り値　TTHEN

ソースコード 41　thenPrint

```
1 int thenPrint(int count);
```

3.4.39　noteqPrint() 関数

■機能　token の一文字目が’¡’ であったとき、次の文字によって token を判定する。

■引数　なし。

■戻り値　各 token 番号

ソースコード 42　noteqPrint

```
1 int noteqPrint(void);
```

3.4.40　grPrint() 関数

■機能　token の一文字目が’¿’ であったとき、次の文字によって token を判定する。

■引数　なし。

■戻り値　各 token 番号

ソースコード 43　grPrint

```
1 int grPrint(void);
```

3.4.41　assignPrint() 関数

■機能　token の一文字目が':' であったとき、次の文字によって token を判定する。

■引数　なし。

■戻り値　各 token 番号

ソースコード 44　assignPrint

```
1 int assignPrint(void);
```

3.4.42　semiPrint() 関数

■機能　token の一文字目が';' であったとき、次の文字によって token を判定する。

■引数　なし。

■戻り値　各 token 番号

ソースコード 45　semiPrint

```
1 int semiPrint(void);
```

3.4.43　endPrint() 関数

■機能　token が end であったとき、前の token によって表示位置を調整する。

■引数　str_atr に格納された文字数

■戻り値　TEND

ソースコード 46　endPrint

```
1 int endPrint(int count);
```

3.5　各関数の外部仕様 (課題 1 まで)

3.5.1　init_scan 関数

■機能　filename のファイルを入力ファイルとしてオープンする。

■引数　開きたいファイルの名前。

■戻り値　正常な場合 0 で、ファイルがオープンできないときは、負の値を返す。

ソースコード 47　init_scan()

```
1 int init_scan(char *filename);
```

3.5.2　scan 関数

■機能　トークンを一つスキャンする関数。

■引数　なし。

■戻り値　次のトークンのコードを返す。トークンコードにない場合は負の値を返す。

ソースコード 48　scan()

```
1 int scan();
```

3.5.3　get_linenum 関数

■機能　行番号を示す関数。

■引数　なし。

■戻り値　scan() で返されたトークンが存在した行の番号を返す．まだ一度も scan() が呼ばれていないときには 0 を返す．

ソースコード 49　get_linenum()

```
1 int get_linenum();
```

3.5.4　end_scan 関数

■機能　init_scan(filename) でオープンしたファイルをクローズする関数。

■引数　なし。

■戻り値　なし。

ソースコード 50　end_scan()

```
1 void end_scan();
```

3.5.5　isChar 関数

■機能　fgetc で収得した文字がアルファベット (a〜z もしくは A〜Z) のいずれかであるか判定する関数。

■引数　判定の対象となる文字。

■戻り値　文字がアルファベットの場合は 1 を返し、文字がアルファベット以外のときは 0 を返す。

ソースコード 51　isChar()

```
1 int isChar(int c);
```

3.5.6　UntilFun 関数

■機能　ある特定の文字まで fgetc で文字を収得する関数。たとえば cbuf で { が得られたとき、} が得られるまで cbuf を進めたいならば、UntilFun('}') とする。

■引数　fgetc で収得したい文字。

■戻り値　なし。

ソースコード 52　UntilFun()

```
1 void UntilFun(int c);
```

3.5.7　UntilComment 関数

■機能　コメントの文字まで fgetc で文字を収得する関数。

■引数　なし。

■戻り値　なし。

ソースコード 53　UntilComment()

```
1 void UntilComment(void);
```

3.5.8　UntilString 関数

■機能　シングルクォーテーションまで fgetc で文字を収得する関数。ただし、シングルクォーテーションが連続のとき (”) は文字列として認めない。

■引数　なし。

■戻り値　なし。

ソースコード 54　UntilString()

```
1 void UntilString(void);
```

3.5.9　init_idtab 関数

■機能　Name のインスタンスのテーブルを初期化する関数。

■引数　なし。

■戻り値　なし。

ソースコード 55　init_idtab()

```
1 void init_idtab();
```

3.5.10　id_countup 関数

■機能　Name のインスタンスを登録してカウントアップする関数。

■引数　収得した Name のトークン。

■戻り値　なし。

ソースコード 56　id_countup()

```
1 void id_countup(char *np);
```

3.5.11　print_idtab 関数

■機能　登録された Name のインスタンスの文字列とカウントを表示する関数。

■引数　なし。

■戻り値　なし。

ソースコード 57　print_idtab()

```
1 void print_idtab();
```

3.5.12　release_idtab 関数

■機能　登録された Name のテーブルの領域を解放する関数。

■引数　なし。

■戻り値　なし。

ソースコード 58　release_idtab()

```
1 void release_idtab();
```

4　テスト情報

4.1　テストデータ

ここでは既に用意されているテストデータについて、ファイル名のみを記述する。

ブラックボックステストとしてのファイルは以下である

- sample11.mpl
- sample12.mpl

ホワイトボックステストとしてのファイルは以下の 76 個である。

- sample2a.mpl
- sample02a.mpl
- sample21.mpl
- sample021.mpl
- sample22.mpl
- sample022.mpl
- sample23.mpl
- sample023.mpl
- sample24.mpl
- sample024.mpl
- sample25.mpl
- sample025.mpl
- sample25t.mpl
- sample26.mpl
- sample026.mpl
- sample27.mpl
- sample28p.mpl
- sample29p.mpl
- test1.mpl
- test2.mpl
- test3.mpl
- test4.mpl
- test5.mpl
- test6.mpl
- test7.mpl
- test8.mpl
- test9.mpl

- test10.mpl
- test11.mpl
- test12.mpl
- test13.mpl
- test14.mpl
- test15.mpl
- test16.mpl
- tb1.mpl
- tb2.mpl
- tb3.mpl
- tb4.mpl
- tb5.mpl
- tb6.mpl
- tb7.mpl
- tb8.mpl
- tb9.mpl
- tb10.mpl
- tb11.mpl
- tb12.mpl
- tb13.mpl
- tb14.mpl
- tb15.mpl
- tb16.mpl
- tb17.mpl
- tb18.mpl
- tb19.mpl
- tb20.mpl
- tb21.mpl
- tb22.mpl
- tb23.mpl
- ta1.mpl
- ta2.mpl
- ta3.mpl
- ta4.mpl
- ta5.mpl
- ta6.mpl
- ta7.mpl
- ta8.mpl
- ta9.mpl
- ta10.mpl
- ta11.mpl
- ta12.mpl
- tt1.mpl
- tt2.mpl
- tt3.mpl
- tt4.mpl
- tt5.mpl
- tt6.mpl
- tt7.mpl
- tm1.mpl
- tm2.mpl
- tm3.mpl
- tm4.mpl

4.2　テスト結果

ここではテストしたすべてのテストデータについて記述する。
以下のようなコマンド実行した。

ソースコード 59　演習室環境での program の実行例

```
$ gcc −o program pprinter−list.c pprinter−list.h ebnf−list.c scan−list.c
$ ./program sample1p.mpl
```

結果は以下のようになった。

ソースコード 60　result

```
1  program sample11;
2      var n , sum , data: integer;
3
4          writeln ( 'input the number of data' );
5          readln ( n );
6          sum := 0;
7          while n> 0 do
8
9              readln ( data );
10             sum := sum + data;
11             n := n − 1
12         end;
```

```
13              writeln ( 'Sum of data = ' , sum )
14          end .
```

プリティプリンタする前の sample1p.mpl は 61 である。

ソースコード 61　sample1p.mpl

```
1   /* プリティプリントする前のリスト*/
2   program sample11;var n,sum,data:integer;begin
3   writeln ('input the number of data') ;readln( n );sum:=0 ;while n>0
4   do
5   begin readln(data);sum:=sum+data;n:=n-1end;writeln('Sum of data = ',sum)end
6   .
```

次に文字と数字の境界部分をテストする。文字の最大数は 1024 であり、数字の最大値は 2,147,483,647 である。

tm1.mpl には a を 1024 個並べたものを書いた。tm2.mpl には a を 1025 個並べたものを書いた。結果は以下の通りになった。

ソースコード 62　a を 1024 個書いたときの結果

```
1    program aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
2           .....
3        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaatm1.mpl;
4         var a: integer;
5         a: integer;
6         begin
7
8         end .
```

ソースコード 63　a を 1025 個書いたときの結果

```
1    program
2   ERROR(2): string is too long
3   Check grammar of 2 line in am2.mpl
```

test3.mpl には a を 2,147,483,647 書いた。test4.mpl には 2,147,483,648 書いた。結果は以下の通りになった。

ソースコード 64　2,147,483,647 書いたときの結果

```
1   program a;
2       var b: integer;
3       a: integer;
4       begin
5           b := 2147483647
6       end .
```

ソースコード 65　2,147,483,648 書いたときの結果

```
1   program a;
2       var b: integer;
3       a: integer;
```

```
4           begin
5               b :=
6   ERROR(4): number is too long
7   Check grammar of 4 line in tm4.mpl
```

続いて、bash ファイル (コード 72) を用いてホワイトボックステストを実行した。以下のコード 66 のようにコマンドを実行した。

ソースコード 66　bash を用いた comand の実行例

```
$ bash comand.sh
```

ソースコード 67　comand.sh の結果 1

```
1  PS> bash comand.sh
2  rm *.gcda *.gcno *.gcov
3  gcc --coverage -o tc id-list.c scan-list.c pprinter-list.c token-list.h ebnf-list.c
4  ./tc sample2a.mpl
5
6   program change;
7       var n , count: integer;
8       begin
9           writeln ( 'please input change' );
10          readln ( n );
11          count := 0;
12          while n> 0 do
13          begin
14              while n>= 10 do
15              begin
16                  while n>= 100 do
17                  begin
18                      while n>= 1000 do
19                      begin
20                          while n>= 10000 do
21                          begin
22                              count := count + 1;
23                              n := n - 10000;
24                          end;
25                          if count> 0 then
26                              writeln ( '10000 yen : ' , count );
27                              count := 0;
28                          if n< 1000 then
29                              break;
30                              count := count + 1;
31                              n := n - 1000;
32                      end;
33                      if count> 0 then
34                          writeln ( ' 1000 yen : ' , count );
35                          count := 0;
36                      if n< 100 then
```

```
                        break;
                        count := count + 1;
                        n := n - 100;
                end;
                if count> 0 then
                    writeln ( ' 100 yen : ' , count );
                    count := 0;
                if n< 10 then
                    break;
                    count := count + 1;
                    n := n - 10;
            end;
            if count> 0 then
                writeln ( ' 10 yen : ' , count );
                count := 0;
            if n< 1 then
                break;
                count := count + 1;
                n := n - 1;
        end;
        if count> 0 then
            writeln ( ' 1 yen : ' , count );
    end .
./tc sample02a.mpl

 program TeaBreak;
    begin
        ;
        ;
        ;
        ;
        ;
        ;
        ;
        ;
        ;
        ;
         writeln ( 'Say "BREAK!!" ' );
        ;
        ;
        ;
        ;
        ;
        ;
        ;
        ;
        ;
        ;
         break
```

```
86        end .
87  ./tc sample21.mpl
88
89   program assigninteger;
90        var n: integer;
91        begin
92             n := 1
93        end .
94  ./tc sample021.mpl
95
96   program assigninteger var
97  ERROR(2): ';' is not found in parse_program
98  Check grammar of 2 line in sample021.mpl
99
100 ./tc sample022.mpl
101
102  program assignboolean;
103       var x y
104 ERROR(2): ':' is not found in variable_declaration1
105 Check grammar of 2 line in sample022.mpl
106
107 ./tc sample22.mpl
108
109  program assignboolean;
110       var x , y: boolean;
111       begin
112            x := true;
113            y := false;
114       end .
115 ./tc sample23.mpl
116
117  program assignchar;
118       var x , y: char;
119       begin
120            x := 'X';
121            y := 'Y'
122       end .
123 ./tc sample023.mpl
124
125  program assignchar;
126       var x , y: char;
127      ;
128
129 ERROR(2): Keyword 'begin' is not found
130 Check grammar of 2 line in sample023.mpl
131
132 ./tc sample024.mpl
133
134  program Write;
```

```
    begin
      ;
        writeln ( 'It' s
ERROR(3): ')' is not found in output_statement
Check grammar of 3 line in sample024.mpl

./tc sample24.mpl

 program Write;
    begin
        writeln ( 'It''s OK?' )
    end .
./tc sample25.mpl

 program ifst;
    var ch: char;
    begin
        readln ( ch );
        if ch = 'a' then
            writeln ( 'It is ''a'' ' )
        else writeln
        ( 'It is not ''a'' ' )
    end .
./tc sample025.mpl

 program ifst;
    var ch: char;
    begin
        readln ( ch );
        if ch = 'a' then
            writeln ( 'It is ''a'' ' );

        else
ERROR(6): Keyword 'end' is not found in compound_statement
Check grammar of 6 line in sample025.mpl

./tc sample025t.mpl

 program ifst;
    var ch: char;
    begin
        readln ( ch );
        if ch = 'a' writeln
ERROR(8): Keyword 'then' is not found in condition_statement
Check grammar of 8 line in sample025t.mpl

./tc sample25t.mpl

 program IfstTC;
```

```
    var ch: char;
    int: integer;
    boolx , booly: boolean;
    begin
        boolx := true;
        booly := false;
        ch := 'a';
        int := 66;
        write ( integer ( ch ) );
        write ( integer ( int ) );
        write ( integer ( boolx ) , integer ( booly ) );
        writeln;
        writeln ( char ( ch ) , char ( int ) );
        writeln ( boolean ( ch ) , integer ( int ) , integer ( boolx ) , integer ( booly ) );
    end .
./tc sample26.mpl

 program whilest;
    var n , i , sum: integer;
    begin
        readln ( n );
        i := n;
        sum := 0;
        while i> 0 do
        begin
            sum := sum + i;
            i := i - 1
        end;
        writeln ( 'Summention of 1 - ' , n , ' is ' , sum )
    end .
./tc sample026.mpl

 program whilest;
    var n , i , sum: integer;
    begin
        readln ( n );
        i := n;
        sum := 0;
        while ( i> 0 ) do
        begin
            sum := sum + i;
            i := i - 1
        end;
        writeln ( 'Summention of 1 - ' , n , ' is ' , sum )
    end .
./tc sample026t.mpl

 program whilest;
    var n , i , sum: integer;
```

```
233     begin
234         readln ( n );
235         i := n;
236         sum := 0;
237         while i> 0
238         begin
239
240 ERROR(9): Keyword 'do' is not found in iteration_statement
241 Check grammar of 9 line in sample026t.mpl
242
243 ./tc sample27.mpl
244
245  program nwhilest;
246     var i , j , k: integer;
247     begin
248         i := 1;
249         while i< 10 do
250         begin
251             j := 1;
252             while j< 10 do
253             begin
254                 k := 1;
255                 while k< 10 do
256                 begin
257                     if ( k div 2 ) * 2 = k then
258                     begin
259                         k := k + 1
260                     end
261                     else
262                     begin
263
264                         k := k + 1
265                     end
266                 end;
267                 j := j + 1;
268             end;
269             i := i + 1
270         end;
271         writeln ( 'All End' )
272     end .
273 ./tc sample28p.mpl
274
275  program sample28p;
276     procedure p;
277     begin
278         writeln ( 'Hello!' )
279     end;
280     procedure q;
281     begin
```

```
282             writeln ( 'Everyone!' )
283         end;
284         begin
285             call p;
286             call q
287         end .
288 ./tc sample028p.mpl
289
290  program sample28p;
291         procedure p;
292         begin
293             writeln ( 'Hello!' )
294         end;
295         procedure q;
296         begin
297             writeln ( 'Everyone!' )
298         end;
299         begin
300             call;
301
302 ERROR(4): Procedure name is not found in call_statement
303 Check grammar of 4 line in sample028p.mpl
304
305 ./tc sample028q.mpl
306
307  program sample28p;
308         procedure p;
309         begin
310             writeln ( 'Hello!' )
311         end;
312         procedure q;
313         begin
314             writeln ( 'Everyone!' )
315         end;
316         begin
317             call p;
318             call q ( a
319         end
320 ERROR(4): ')' is not found in call_statement
321 Check grammar of 4 line in sample028q.mpl
322
323 ./tc sample028r.mpl
324
325  program sample28p;
326         procedure p;
327         begin
328             writeln ( 'Hello!' )
329         end;
330         procedure q;
```

```
331     begin
332         writeln ( 'Everyone!' )
333     end;
334     begin
335         call p;
336         call q ( ( a )
337     end
338 ERROR(4): ')' is not found in call_statement
339 Check grammar of 4 line in sample028r.mpl
340 
341 ./tc sample29p.mpl
342 
343  program sample29;
344     var unused1: integer;
345     UnusedArrayForTest: array [ 200 ] of char;
346     procedure gcmlcm ( m , n , gc , lc: integer );
347     var a , b , r: integer;
348     begin
349         a := m;
350         b := n;
351         while b<> 0 do
352         begin
353             r := a - ( a div b ) * b;
354             a := b;
355             b := r
356         end;
357         gc := a;
358         lc := ( m div gc ) * n
359     end;
360     procedure abs ( a , b: integer );
361     begin
362         if a< 0 then
363             b := - a
364         else b
365         := a
366     end;
367     procedure gcm ( a , b , gc: integer );
368     var lc , aa , bb: integer;
369     begin
370         if ( a = 0 ) or ( b = 0 ) then
371             gc := 1
372         else
373         begin
374 
375             call abs ( a , aa );
376             call abs ( b , bb );
377             call gcmlcm ( aa , bb , gc , lc )
378         end
379     end;
```

```
380     procedure lcm ( a , b , lc: integer );
381     var gc , aa , bb: integer;
382     begin
383         if ( a = 0 ) or ( b = 0 ) then
384             lc := 1
385         else
386         begin
387
388             call abs ( a , aa );
389             call abs ( b , bb );
390             call gcmlcm ( aa , bb , gc , lc )
391         end
392     end;
393     var unusedchar: char;
394     procedure reduce ( a1 , a2: integer );
395     var gc: integer;
396     begin
397         if a1 = 0 then
398         begin
399             a2 := 1;
400             return
401         end;
402         if a2 = 0 then
403         begin
404             a1 := 1;
405             return
406         end;
407         if a2< 0 then
408         begin
409             a1 := - a1;
410             a2 := - a2
411         end;
412         call gcm ( a1 , a2 , gc );
413         a1 := a1 div gc;
414         a2 := a2 div gc
415     end;
416     procedure sum ( x1 , x2 , y1 , y2: integer );
417     var lc , y11: integer;
418     begin
419         call lcm ( x2 , y2 , lc );
420         x1 := x1 * ( lc div x2 );
421         y11 := y1 * ( lc div y2 );
422         x1 := x1 + y11;
423         x2 := lc;
424         call reduce ( x1 , x2 )
425     end;
426     procedure sub ( x1 , x2 , y1 , y2: integer );
427     var lc , y11: integer;
428     begin
```

```
429             call sum ( x1 , x2 , - y1 , y2 )
430         end;
431         procedure mult ( x1 , x2 , y1 , y2: integer );
432         var gc , y22 , y11: integer;
433         begin
434             call gcm ( x1 , y2 , gc );
435             x1 := x1 div gc;
436             y22 := y2 div gc;
437             call gcm ( x2 , y1 , gc );
438             x2 := x2 div gc;
439             y11 := y1 div gc;
440             x1 := x1 * y11;
441             x2 := x2 * y22;
442             call reduce ( x1 , x2 )
443         end;
444         procedure divide ( x1 , x2 , y1 , y2: integer );
445         begin
446             call mult ( x1 , x2 , y2 , y1 )
447         end;
448         var unusedarray: array [ 100 ] of char;
449         procedure printfinal ( a , b: integer );
450         begin
451             if a = 0 then
452                 writeln ( 'Final Result =' , a )
453             else if b = 1 then
454                 writeln ( 'Final Result =' , a )
455             else writeln
456             ( 'Final Result =' , a , '/' , b )
457         end;
458         procedure printtemp ( a , b: integer );
459         begin
460             if a = 0 then
461                 writeln ( 'Temporary Result =' , a )
462             else if b = 1 then
463                 writeln ( 'Temporary Result =' , a )
464             else writeln
465             ( 'Temporary Result =' , a , '/' , b )
466         end;
467         var x1 , x2 , y1 , y2: integer;
468         var com: char;
469         endflag: boolean;
470         begin
471             writeln ( ' *** Calculator -- h for help ***' );
472             x1 := 0;
473             x2 := 1;
474             endflag := false;
475             while not endflag do
476             begin
477                 writeln ( ' Please input command :' );
```

```
478                 readln ( com , y1 );
479                 y2 := 1;
480                 if ( com = 'c' ) or ( com = 'C' ) then
481                 begin
482                     x1 := y1;
483                     x2 := y2
484                 end
485                 else if com = '+' then
486                     call sum ( x1 , x2 , y1 , y2 )
487                 else if com = '-' then
488                     call sub ( x1 , x2 , y1 , y2 )
489                 else if com = '*' then
490                     call mult ( x1 , x2 , y1 , y2 )
491                 else if com = '/' then
492                     call divide ( x1 , x2 , y1 , y2 )
493                 else if ( com = 'o' ) or ( com = 'O' ) then
494                     endflag := true
495                 else
496                 begin
497
498                     writeln;
499                     writeln ( 'Calculator Usage:' );
500                     writeln ( ' c number : clear & set it' );
501                     writeln ( ' + number : add it' );
502                     writeln ( ' - number : subtract it' );
503                     writeln ( ' * number : multiply it' );
504                     writeln ( ' / number : divide by it' );
505                     writeln ( ' o : off(terminate execution)' );
506                     writeln
507                 end;
508                 if endflag then
509                     call printfinal ( x1 , x2 )
510                 else call
511                 printtemp ( x1 , x2 )
512             end
513         end .
514 ./tc sample029p.mpl
515
516  program sample29;
517         var com: char;
518         endflag: boolean;
519         begin
520             readln ( ,
521 ERROR(5): Variable name is not found in variable
522 Check grammar of 5 line in sample029p.mpl
523
524 ./tc sample029q.mpl
525
526  program sample29;
```

```
527       var x1 , x2 , y1 , y2: integer;
528       var com: char;
529       endflag: boolean;
530       begin
531           readln ( com , y1;
532
533 ERROR(6): ')' is not found in input_statement
534 Check grammar of 6 line in sample029q.mpl
535
536 ./tc sample029r.mpl
537 File sample029r.mpl can not open.
538 ./tc sample029s.mpl
539
540  program sample29;
541       var x1 , x2 , y1 , y2: integer;
542       var com: char;
543       endflag: boolean;
544       begin
545           writeln ( term: )
546 ERROR(6): Number is not found in output_format
547 Check grammar of 6 line in sample029s.mpl
548
549 ./tc sample029t.mpl
550
551  program sample29;
552       var x1 , x2 , y1 , y2: integer;
553       var com: char;
554       endflag: boolean;
555       begin
556           writeln ( term: 1 )
557 ERROR(7): Keyword 'end' is not found in compound_statement
558 Check grammar of 7 line in sample029t.mpl
559
560 test1.mpl
561  aaa
562 ERROR(1): Keyword 'program' is not found
563 Check grammar of 1 line in test1.mpl
564
565 test2.mpl
566
567  program
568 ERROR(1): Program name is not found
569 Check grammar of 1 line in test2.mpl
570
571 test3.mpl
572
573  program aa
574 ERROR(1): ';' is not found in parse_program
575 Check grammar of 1 line in test3.mpl
```

```
576
577 test4.mpl
578
579  program aa;
580       var
581 ERROR(1): Variable name is not found in variable_names1
582 Check grammar of 1 line in test4.mpl
583
584 test5.mpl
585
586  program aa;
587       var 1
588 ERROR(1): Variable name is not found in variable_names1
589 Check grammar of 1 line in test5.mpl
590
591 test6.mpl
592
593  program aa;
594       var a
595 ERROR(1): ':' is not found in variable_declaration1
596 Check grammar of 1 line in test6.mpl
597
598 test7.mpl
599
600  program aa;
601       var a:
602 ERROR(1): Type is not found in type
603 Check grammar of 1 line in test7.mpl
604
605 test8.mpl
606
607  program aa;
608       var a: integer
609 ERROR(1): ';' is not found in variable_declaration
610 Check grammar of 1 line in test8.mpl
611
612 test9.mpl
613
614  program aa;
615       var a: integer;
616       a
617 ERROR(1): ':' is not found in variable_declaration2
618 Check grammar of 1 line in test9.mpl
619
620 test10.mpl
621
622  program aa;
623       var a: integer;
624       a:
```

```
625 ERROR(1): Type is not found in type
626 Check grammar of 1 line in test10.mpl
627
628 test11.mpl
629
630  program aa;
631       var a: integer;
632       a: integer
633 ERROR(1): ';' is not found in variable_declaration
634 Check grammar of 1 line in test11.mpl
635
636 test12.mpl
637
638  program aa;
639       var a: integer;
640       a: integer;
641
642       program
643 ERROR(2): Keyword 'begin' is not found
644 Check grammar of 2 line in test12.mpl
645
646 test13.mpl
647
648  program aa;
649       var a: integer;
650       a: integer;
651       begin
652
653 ERROR(2): Keyword 'end' is not found in compound_statement
654 Check grammar of 2 line in test13.mpl
655
656 test14.mpl
657
658  program aa;
659       var a: integer;
660       a: integer;
661       begin
662
663       end
664 ERROR(2): '.' is not found at the end of program
665 Check grammar of 2 line in test14.mpl
666
667 test15.mpl
668
669  program aa;
670       var a: integer;
671       a: integer;
672       begin
673
```

```
674         end .
675 test16.mpl
676
677  program aa;
678         var a: integer;
679         a ,:
680 ERROR(1): Variable name is not found in variable_names2
681 Check grammar of 1 line in test16.mpl
682
683 tb1.mpl
684
685  program aa;
686         procedure
687 ERROR(1): Procedure_name is not found in subprogram_declaration
688 Check grammar of 1 line in tb1.mpl
689
690 tb2.mpl
691
692  program aa;
693         procedure a
694 ERROR(1): ';' is not found in subprogram_declaration
695 Check grammar of 1 line in tb2.mpl
696
697 tb3.mpl
698
699  program aa;
700         procedure a (
701 ERROR(1): Variable name is not found in variable_names1
702 Check grammar of 1 line in tb3.mpl
703
704 tb4.mpl
705
706  program aa;
707         procedure a ( a
708 ERROR(1): ':' is not found in formal_parameters
709 Check grammar of 1 line in tb4.mpl
710
711 tb5.mpl
712
713  program aa;
714         procedure a ( a:
715 ERROR(1): Type is not found in type
716 Check grammar of 1 line in tb5.mpl
717
718 tb6.mpl
719
720  program aa;
721         procedure a ( a: boolean
722 ERROR(1): ')' is not found in formal_parameters
```

```
Check grammar of 1 line in tb6.mpl

tb7.mpl

 program aa;
      procedure a ( a: char
ERROR(1): ')' is not found in formal_parameters
Check grammar of 1 line in tb7.mpl

tb8.mpl

 program aa;
      procedure a ( a: char;

ERROR(1): Variable name is not found in variable_names1
Check grammar of 1 line in tb8.mpl

tb9.mpl

 program aa;
      procedure a ( a: char;
      a
ERROR(1): ':' is not found in formal_parameters in in formal_parameters
Check grammar of 1 line in tb9.mpl

tb10.mpl

 program aa;
      procedure a ( a: char;
      a:
ERROR(1): Standard type is not found in standard_type
Check grammar of 1 line in tb10.mpl

tb11.mpl

 program aa;
      procedure a ( a: char;
      a: integer
ERROR(1): ')' is not found in formal_parameters
Check grammar of 1 line in tb11.mpl

tb12.mpl

 program aa;
      procedure a ( a: char;
      a: integer )
ERROR(1): ';' is not found in subprogram_declaration
Check grammar of 1 line in tb12.mpl

```

```
tb13.mpl

 program aa;
      procedure a ( a: char;
      a: integer );

ERROR(1): Keyword 'begin' is not found
Check grammar of 1 line in tb13.mpl

tb14.mpl

 program aa;
      procedure a ( a: char;
      a: integer );
      var
ERROR(1): Variable name is not found in variable_names1
Check grammar of 1 line in tb14.mpl

tb15.mpl

 program aa;
      procedure a ( a: char;
      a: integer );
      var b: integer;
      begin

ERROR(1): Keyword 'end' is not found in compound_statement
Check grammar of 1 line in tb15.mpl

tb16.mpl

 program aa;
      procedure a ( a: char;
      a: integer );
      var b: integer;
      begin

      end;

ERROR(1): Keyword 'begin' is not found
Check grammar of 1 line in tb16.mpl

tb17.mpl

 program aa;
      procedure a ( a: char;
      a: integer );
      var b: integer;
      begin
```

```
821
822         end;
823         begin
824
825         end .
826 tb18.mpl
827
828  program aa;
829         procedure a:
830 ERROR(1): ';' is not found in subprogram_declaration
831 Check grammar of 1 line in tb18.mpl
832
833 tb19.mpl
834
835  program aa;
836         procedure a ( a: char;
837         a: integer );
838         var b: integer;
839         begin
840
841         end:
842 ERROR(1): ';' is not found in subprogram_declaration
843 Check grammar of 1 line in tb19.mpl
844
845 tb20.mpl
846
847  program aa;
848         var a: integer;
849         a: integer;
850         begin
851             hensyu [ ( term ]
852 ERROR(2): ')' is not found in factor
853 Check grammar of 2 line in tb20.mpl
854
855 tb21.mpl
856
857  program aa;
858         var a: integer;
859         a: integer;
860         begin
861             hensyu [ integer ( term ]
862 ERROR(2): ')' is not found in factor
863 Check grammar of 2 line in tb21.mpl
864
865 tb22.mpl
866
867  program aa;
868         var a: integer;
869         a: integer;
```

```
870         begin
871             hensyu [ integer 1
872 ERROR(2): '(' is not found in factor
873 Check grammar of 2 line in tb22.mpl
874
875 tb23.mpl
876
877  program aa;
878         var a: integer;
879         a: integer;
880         begin
881             hensyu [ .
882 ERROR(2): Factor is not found in factor
883 Check grammar of 2 line in tb23.mpl
884 tt1.mpl
885
886  program aa;
887         procedure a ( a: array
888 ERROR(1): '[' is not found in array_type
889 Check grammar of 1 line in tt1.mpl
890
891 tt2.mpl
892
893  program aa;
894         procedure a ( a: array [
895 ERROR(1): Number is not found in array_type
896 Check grammar of 1 line in tt2.mpl
897
898 tt3.mpl
899
900  program aa;
901         procedure a ( a: array [ 1
902 ERROR(1): ']' is not found in array_type
903 Check grammar of 1 line in tt3.mpl
904
905 tt4.mpl
906
907  program aa;
908         procedure a ( a: array [ 1 ]
909 ERROR(1): Keyword 'of' is not found in array_type
910 Check grammar of 1 line in tt4.mpl
911
912 tt5.mpl
913
914  program aa;
915         procedure a ( a: array [ 1 ] of
916 ERROR(1): Standard type is not found in standard_type
917 Check grammar of 1 line in tt5.mpl
918
```

```
tt6.mpl

 program aa;
      procedure a ( a: array [ 1 ] of integer
ERROR(1): ')' is not found in formal_parameters
Check grammar of 1 line in tt6.mpl

tt7.mpl

 program aa;
      procedure a ( a: array [ 1 ] of integer )
ERROR(1): ';' is not found in subprogram_declaration
Check grammar of 1 line in tt7.mpl

ta1.mpl

 program aa;
      var a: integer;
      a: integer;
      begin
          hensyu
ERROR(2): ':=' is not found in return_statement
Check grammar of 2 line in ta1.mpl

ta2.mpl

 program aa;
      var a: integer;
      a: integer;
      begin
          hensyu [
ERROR(2): Factor is not found in factor
Check grammar of 2 line in ta2.mpl

ta3.mpl

 program aa;
      var a: integer;
      a: integer;
      begin
          hensyu [ +
ERROR(2): Factor is not found in factor
Check grammar of 2 line in ta3.mpl

ta4.mpl

 program aa;
      var a: integer;
      a: integer;
```

```
968       begin
969           hensyu [ −
970 ERROR(2): Factor is not found in factor
971 Check grammar of 2 line in ta4.mpl
972
973 ta5.mpl
974
975  program aa;
976       var a: integer;
977       a: integer;
978       begin
979           hensyu [ − term
980 ERROR(2): ']' is not found in variable
981 Check grammar of 2 line in ta5.mpl
982
983 ta6.mpl
984
985  program aa;
986       var a: integer;
987       a: integer;
988       begin
989           hensyu [ − term ]
990 ERROR(2): ':=' is not found in return_statement
991 Check grammar of 2 line in ta6.mpl
992
993 ta7.mpl
994
995  program aa;
996       var a: integer;
997       a: integer;
998       begin
999           hensyu [ − term ] :=
1000 ERROR(2): Factor is not found in factor
1001 Check grammar of 2 line in ta7.mpl
1002
1003 ta8.mpl
1004
1005  program aa;
1006       var a: integer;
1007       a: integer;
1008       begin
1009           hensyu [ − term ] := false
1010 ERROR(2): Keyword 'end' is not found in compound_statement
1011 Check grammar of 2 line in ta8.mpl
1012
1013 ta9.mpl
1014
1015  program aa;
1016       var a: integer;
```

```
1017        a: integer;
1018        begin
1019            hensyu [ - term ] := true
1020  ERROR(2): Keyword 'end' is not found in compound_statement
1021  Check grammar of 2 line in ta9.mpl
1022
1023  ta10.mpl
1024
1025   program aa;
1026        var a: integer;
1027        a: integer;
1028        begin
1029            hensyu [ - term ] := 'a'
1030  ERROR(2): Keyword 'end' is not found in compound_statement
1031  Check grammar of 2 line in ta10.mpl
1032
1033  ta11.mpl
1034
1035   program aa;
1036        var a: integer;
1037        a: integer;
1038        begin
1039            readln
1040  ERROR(2): Keyword 'end' is not found in compound_statement
1041  Check grammar of 2 line in ta11.mpl
1042
1043  ta12.mpl
1044
1045   program aa;
1046        var a: integer;
1047        a: integer;
1048        begin
1049            writeln
1050  ERROR(2): Keyword 'end' is not found in compound_statement
1051  Check grammar of 2 line in ta12.mpl
1052  tm1.mpl
1053
1054   program aaaaaaaaaaaaaaaaaaaaaaaaa
1055  ...
1056  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaatm1.mpl;
1057        var a: integer;
1058        a: integer;
1059        begin
1060
1061        end .
1062  tm2.mpl
1063
1064   program
1065  ERROR(2): string is too long
```

```
Check grammar of 2 line in am2.mpl

tm3.mpl

 program a;
      var b: integer;
      a: integer;
      begin
           b := 2147483647
      end .
tm4.mpl

 program a;
      var b: integer;
      a: integer;
      begin
           b :=
ERROR(4): number is too long
Check grammar of 4 line in tm4.mpl

gcov −b ebnf−list.gcda
File 'ebnf-list.c'
Lines executed:100.00% of 312
Branches executed:100.00% of 186
Taken at least once:98.39% of 186
Calls executed:100.00% of 164
Creating 'ebnf-list.c.gcov'
```

4.3　テストデータの十分性

ここではそれだけのテストでどの程度バグがないことが保証できるかを記述する
まず、コード 60 の結果をみたとき、プリティプリンタする前のプログラム (コード 61) がプリティプリンタされていることがわかる。コメントアウトや余分な空白と改行は省略され、begin や then で段落上げがあり、else や end で段落下げが行われていることがわかる。
文字や数字の最大値と一致している、またはそれ以下のときはエラーは出ないが、最大を１つでも過ぎるとエラーが表示されていることも確認できてた。
また、文法的誤りがあるサンプルコードに０があるもの (sample021.mpl,sample022.mpl) にはすべてエラーが表示され、どのファイルの何行目どのエラーが出たか表示ができていることも分かる。このレポートには色まで再現ができていないが、実際に実行すると、エラーは赤文字で表示され、強調され見やすくなっていることもわかる。
このように、プリティプリンタしたいファイルをコマンドライン引数にしてやると、文法的誤りが有ればエラーを表示し、なければ、シンプルで見やすいようインデントされたコードにして表示できていることが確認できた。

続いて、コード 72 を実行したときの結果 (コード 67) みたとき、

- Lines executed:100.00% of 312
- Branches executed:100.00% of 186
- Taken at least once:98.39% of 186
- Calls executed:100.00% of 164

であった。それぞれ以下のような意味を示す。[2]

- Lines executed 実行ラインをどれだけ通過したかを表す。C0 カバレッジ
- Branches executed 条件分岐行をどれだけ実行したか。C1 カバレッジ
- Taken at least once 各条件分岐の組合せを 1 回は通過したか。C1 カバレッジ
- Calls executed 対象関数内で別関数を呼んだかどうか。S0 カバレッジ

c0 カバレッジと s0 カバレッジの網羅率は 100% であった。C1 カバレッジでは全部の各条件分岐の網羅率は 98.39% であったが、全条件分岐の網羅率は 100% となり、テストが十分できたことが評価できる。
以上のブラックボックス、ホワイトボックスのテストよりバグのないプログラムが書けたと考えられる。

5　事前計画と実際の進捗状況

5.1　事前計画

事前計画は 1 のようになった。

表 1　事前作業計画

開始予定日	終了予定日	見積もり時間	番号	作業内容
11/07	11/07	1	(a)	スケジュールを立てる
11/07	11/07	0.5	(b-1)	配布された資料を読み直す
11/07	11/07	0.5	(b-2)	配布されたプログラムを読む
11/07	11/07	1	(b-3)	コンパイラのテキスト (プログラム) を読む
11/14	11/14	5	(c)	字句解析系の概略設計
11/14	11/14	2	(e-1-1)	ブラックボックステスト用プログラムの作成
11/14	11/14	5	(d-4)	解析器の作成
11/14	11/14	1	(e-1-2)	バグがない場合の想定テスト結果の準備
11/21	11/21	1	(d-3)	カウントした結果の出力部分の作成
11/21	11/21	0.5	(e-2-1)	カバレッジレベルの決定
11/21	11/21	2	(e-2-2)	ホワイトボックステスト用プログラムの作成
11/21	11/21	1	(e-2-3)	バグがない場合の想定テスト結果の準備
11/25	11/25	8	(f)	テストとデバッグを行う
11/25	11/25	1	(g-1)	作成したプログラムの設計情報を書く
11/25	11/25	1	(g-2)	テスト情報を書く
11/27	11/27	1	(g-3)	事前計画と実際の進捗状況を書く
11/27	11/27	5	(h)	プログラムとレポートの提出

5.2　事前計画の立て方についての前課題からの改善点

前回は提出直前になって焦り、テストするのに十分な時間がとれなかった。直前になって焦らないよう計画的に進めるよう改善した。

5.3　実際の進捗状況

実際の計画時間は表 2 のようになった。

表 2　事前作業計画

開始予定日	終了予定日	計画時間	番号	終了日	実際の時間
11/07	11/07	1	(a)	11/03	0.5
11/07	11/07	0.5	(b-1)	11/11	1
11/07	11/07	0.5	(b-2)	11/11	1
11/07	11/07	1	(b-3)	11/11	1
11/14	11/14	5	(c)	11/11	3
11/14	11/14	2	(e-1-1)	11/11	1
11/14	11/14	5	(d-4)	11/16	1
11/14	11/14	1	(e-1-2)	11/16	5
11/21	11/21	1	(d-3)	11/16	1
11/21	11/21	0.5	(e-2-1)	11/16	1
11/21	11/21	2	(e-2-2)	11/16	1
11/21	11/21	1	(e-2-3)	11/16	1
11/25	11/25	8	(f)	11/17	5
11/25	11/25	1	(g-1)	11/18	1
11/25	11/25	1	(g-2)	11/18	1
11/27	11/27	1	(g-3)	11/18	1
11/27	11/27	5	(h)	11/28	10

5.4　当初の事前計画と実際の進捗との差の原因

表 2 より若干の進行に差があった。学校の行事 (学園祭) もあり、１１月中盤は思うように進まないこともあった。なぜ、見積もりと実際の作業に差が生まれたのか。データの根拠がない場合は楽観的になりがちで、多くの人が見積もりを小さい範囲にしてしまうという DeMarco - Glass の法則がある。[3]

6　ソースコード

ソースコード 68　pprinter-list.c

```
1  #include "pprinter-list.h"
2
3  /* keyword list */
4  struct KEY key[KEYWORDSIZE] = {
5          {"and", TAND},
6          {"array", TARRAY},
7          {"begin", TBEGIN},
8          {"boolean", TBOOLEAN},
9          {"break", TBREAK},
10         {"call", TCALL},
11         {"char", TCHAR},
12         {"div", TDIV},
13         {"do", TDO},
14         {"else", TELSE},
15         {"end", TEND},
16         {"false", TFALSE},
17         {"if", TIF},
18         {"integer", TINTEGER},
19         {"not", TNOT},
20         {"of", TOF},
21         {"or", TOR},
22         {"procedure", TPROCEDURE},
23         {"program", TPROGRAM},
24         {"read", TREAD},
25         {"readln", TREADLN},
26         {"return", TRETURN},
27         {"then", TTHEN},
28         {"true", TTRUE},
29         {"var", TVAR},
30         {"while", TWHILE},
31         {"write", TWRITE},
32         {"writeln", TWRITELN}};
33
34 /* Token counter */
35 int numtoken[NUMOFTOKEN + 1] = {0};
36
37 /* string of each token */
38 char *tokenstr[NUMOFTOKEN + 1] = {
39         "",
40         "NAME", "program", "var", "array", "of", "begin", "end", "if", "then",
41         "else", "procedure", "return", "call", "while", "do", "not", "or",
42         "div", "and", "char", "integer", "boolean", "readln", "writeln", "true",
43         "false", "NUMBER", "STRING", "+", "-", "*", "=", "<>", "<", "<=", ">",
44         ">=", "(", ")", "[", "]", ":=", ".", ",", ":", ";", "read", "write", "break"};
```

```
45
46 char filename[FILENAME_MAX];
47
48 int main(int nc, char *np[])
49 {
50         int token, i;
51
52         if (nc < 2)
53         {
54                 printf("File name id not given.\n");
55                 return 0;
56         }
57         if (init_scan(np[1]) < 0)
58         {
59                 printf("File %s can not open.\n", np[1]);
60                 return 0;
61         }
62         strcat(filename, np[1]);
63         token = scan();
64         set_token(token);
65         parse_program();
66
67         end_scan();
68         return 0;
69 }
70
71 void error(char *mes)
72 {
73         int line = get_linenum();
74         printf("\x1b[31m");
75         printf("\nERROR(%d): %s", line, mes);
76         printf("Check grammar of %d line in %s\n\n", line, filename);
77         printf("\x1b[0m");
78         end_scan();
79         exit(1);
80 }
```

ソースコード 69　pprinter-list.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAXSTRSIZE 1024
#define MAXINTSIZE 2147483647

/* Token */
#define TNAME 1  /* Name : Alphabet { Alphabet | Digit } */
#define TPROGRAM 2 /* program : Keyword */
#define TVAR 3  /* var : Keyword */
#define TARRAY 4 /* array : Keyword */
#define TOF 5  /* of : Keyword */
#define TBEGIN 6 /* begin : Keyword */
#define TEND 7  /* end : Keyword */
#define TIF 8  /* if : Keyword */
#define TTHEN 9  /* then : Keyword */
#define TELSE 10 /* else : Keyword */
#define TPROCEDURE 11 /* procedure : Keyword */
#define TRETURN 12 /* return : Keyword */
#define TCALL 13 /* call : Keyword */
#define TWHILE 14 /* while : Keyword */
#define TDO 15  /* do : Keyword */
#define TNOT 16  /* not : Keyword */
#define TOR 17  /* or : Keyword */
#define TDIV 18  /* div : Keyword */
#define TAND 19  /* and : Keyword */
#define TCHAR 20 /* char : Keyword */
#define TINTEGER 21 /* integer : Keyword */
#define TBOOLEAN 22 /* boolean : Keyword */
#define TREADLN 23 /* readln : Keyword */
#define TWRITELN 24 /* writeln : Keyword */
#define TTRUE 25 /* true : Keyword */
#define TFALSE 26 /* false : Keyword */
#define TNUMBER 27 /* unsigned integer */
#define TSTRING 28 /* String */
#define TPLUS 29 /* + : symbol */
#define TMINUS 30 /* - : symbol */
#define TSTAR 31 /* * : symbol */
#define TEQUAL 32 /* = : symbol */
#define TNOTEQ 33 /* <> : symbol */
#define TLE 34  /* < : symbol */
#define TLEEQ 35 /* <= : symbol */
#define TGR 36  /* > : symbol */
#define TGREQ 37 /* >= : symbol */
#define TLPAREN 38 /* ( : symbol */
```

```
48 #define TRPAREN 39 /* ) : symbol */
49 #define TLSQPAREN 40 /* [ : symbol */
50 #define TRSQPAREN 41 /* ] : symbol */
51 #define TASSIGN 42 /* := : symbol */
52 #define TDOT 43  /* . : symbol */
53 #define TCOMMA 44 /* , : symbol */
54 #define TCOLON 45 /* : : symbol */
55 #define TSEMI 46 /* ; : symbol */
56 #define TREAD 47 /* read : Keyword */
57 #define TWRITE 48 /* write : Keyword */
58 #define TBREAK 49 /* break : Keyword */
59
60 #define NUMOFTOKEN 49
61
62 /* token-list.c */
63
64 #define KEYWORDSIZE 28
65
66 extern struct KEY
67 {
68         char *keyword;
69         int keytoken;
70 } key[KEYWORDSIZE];
71
72 extern void error(char *mes);
73 extern char *tokenstr[NUMOFTOKEN + 1];
74 extern void init_idtab();
75 extern void id_countup(char *np);
76 extern void print_idtab();
77 extern void release_idtab();
78
79 /* list-scan.c */
80 extern int init_scan(char *filename);
81 extern int scan(void);
82 extern int num_attr;
83 extern char string_attr[MAXSTRSIZE];
84 extern int get_linenum(void);
85 extern void end_scan(void);
86 extern int isNumber(int c);
87 extern int isChar(int c);
88 extern void UntilFun(int c);
89 extern void UntilComment(void);
90 extern void UntilString(void);
91 extern void print_indent(void);
92 extern void print_tab(void);
93 extern void init_string_atr(int count);
94 extern int programPrint(int count);
95 extern int beginPrint(int count);
96 extern int endPrint(int count);
```

```
97  extern int ifPrint(int count);
98  extern int elsePrint(int count);
99  extern int thenPrint(int count);
100 extern int noteqPrint(void);
101 extern int grPrint(void);
102 extern int assignPrint(void);
103 extern int semiPrint(void);
104
105 /* pprinter-list.c */
106 extern void error(char *mes);
107
108 /* ebnf-list.c */
109 extern void set_token(int token);
110 extern int parse_program();
111 extern int parse_block();
112 extern int parse_variable_declaration();
113 extern int parse_variable_names();
114 extern int parse_variable_name();
115 extern int parse_type();
116 extern int parse_standard_type();
117 extern int parse_array_type();
118 extern int parse_subprogram_declaration();
119 extern int parse_procedure_name();
120 extern int parse_formal_parameters();
121 extern int parse_compound_statement();
122 extern int parse_statement();
123 extern int parse_condition_statement();
124 extern int parse_iteration_statement();
125 extern int parse_exit_statement();
126 extern int parse_call_statement();
127 extern int parse_expressions();
128 extern int parse_return_statement();
129 extern int parse_assignment_statement();
130 extern int parse_left_part();
131 extern int parse_variable();
132 extern int parse_expression();
133 extern int parse_simple_expression();
134 extern int parse_term();
135 extern int parse_factor();
136 extern int parse_constant();
137 extern int parse_multiplicative_operator();
138 extern int parse_additive_operator();
139 extern int parse_relational_operator();
140 extern int parse_input_statement();
141 extern int parse_output_statement();
142 extern int parse_output_format();
143 extern int parse_empty_statement();
```

ソースコード 70 scan-list.c

```
#include "pprinter-list.h"
#define ERROR (−1)
#define EOFCODE (−2)
int cbuf, num_line = 1;
int num_indent = 1;
int num_attr;
int num_then = 0;
enum PreState
{
        OTEHER,
        SEMI,
        THEN,
        ELSE
};
enum PreState prestate = OTEHER;
char string_attr[MAXSTRSIZE];
FILE *fp = NULL;

int init_scan(char *filename)
{ /* open file if it suceed return 0 and if not return -1 */
        if ((fp = fopen(filename, "r+")) == NULL)
        {
                return ERROR;
        }
        else
        {
                cbuf = fgetc(fp);
                return 0;
        }
}

int scan(void)
{
        int isSeparator = 1;
        while (isSeparator)
        { // if separator, skip read
                switch (cbuf)
                {
                case '\r': // end of line
                        cbuf = fgetc(fp);
                        if (cbuf == '\n')
                        {
                                cbuf = fgetc(fp);
                        }
                        num_line++;
                        break;
                case '\n': // end of line Unix,Mac
```

```
48                      cbuf = fgetc(fp);
49                      if (cbuf == '\r')
50                      {
51                              cbuf = fgetc(fp);
52                      }
53                      num_line++;
54                      break;
55              case ' ': // space
56              case '\t': // tab
57                      cbuf = fgetc(fp);
58                      break;
59              case '{': // coment {...}
60                      UntilFun('}');
61                      break;
62              case '/':
63                      cbuf = fgetc(fp);
64                      if (cbuf == '*')
65                      {
66                              UntilComment();
67                              break;
68                      }
69                      else
70                      {
71                              error("/ is not undeclared\n");
72                              return ERROR;
73                      }
74                      break;
75              case '\'':
76                      UntilString();
77                      return TSTRING;
78                      break;
79              default:
80                      isSeparator = 0;
81                      break;
82              }
83      }
84      int count = 0;
85      if (isChar(cbuf))
86      { // if char, read them by end
87              int i, j = 0;
88              for (i = 0; (isChar(cbuf) + isNumber(cbuf)) >= 1; i++)
89              {
90                      string_attr[i] = cbuf;
91                      if (i >= MAXSTRSIZE)
92                      {
93                              error("string is too long\n");
94                              return ERROR;
95                      }
96                      cbuf = fgetc(fp);
```

```
97                      count++;
98              }
99              // cbuf = fgetc(fp);
100             // id_countup(string_attr);
101             for (j = 0; j <= NUMOFTOKEN; j++)
102             { // check whether keyword or name
103                     if (strcmp(string_attr, key[j].keyword) == 0)
104                     {
105                             if (key[j].keytoken == TPROGRAM)
106                             {
107                                     return programPrint(count);
108                             }
109                             else if (key[j].keytoken == TBEGIN)
110                             { // if begin, print next line
111                                     return beginPrint(count);
112                             }
113                             else if (key[j].keytoken == TEND)
114                             { // if end, print next line
115                                     return endPrint(count);
116                             }
117                             else if (key[j].keytoken == TIF)
118                             { // if if, print next line
119                                     return ifPrint(count);
120                             }
121                             else if (key[j].keytoken == TELSE)
122                             { // if else, print next line
123                                     return elsePrint(count);
124                             }
125                             else if (key[j].keytoken == TTHEN)
126                             { // if then, print next line
127                                     return thenPrint(count);
128                             }
129                             else
130                             {
131                                     printf(" %s", string_attr);
132                                     init_string_atr(count);
133                                     return key[j].keytoken; // retrun keyword
134                             }
135                     }
136             }
137             printf(" %s", string_attr);
138             init_string_atr(count);
139             return TNAME; // return name
140     }
141     else if (isNumber(cbuf))
142     { // if number, read them by end
143             int i = 0;
144             num_attr = 0;
145             for (i = 0; isNumber(cbuf) > 0; i++)
```

```
146             {
147                     num_attr = num_attr * 10 + (cbuf - 48);
148                     cbuf = fgetc(fp);
149                     if ((num_attr > MAXINTSIZE) || (num_attr < 0))
150                     {
151                             error("number is too long\n");
152                             return ERROR;
153                     }
154             }
155             // cbuf = fgetc(fp);
156             printf(" %d", num_attr);
157             return TNUMBER;
158     }
159     else
160     {
161             int i = TPLUS;
162             for (i = TPLUS; i <= TSEMI; i++)
163             {
164                     if (cbuf == tokenstr[i][0])
165                     {
166                             if (i == TNOTEQ)
167                             { // if cbuf is '<' check whether '<>' , '<=' or '<'
168                                     return noteqPrint();
169                             }
170                             else if (i == TGR)
171                             { // if cbuf is '>' check whether '>=' or '>'
172                                     return grPrint();
173                             }
174                             else if (i == TASSIGN)
175                             { // if cbuf is ':' check whether ':=' or ':'
176                                     return assignPrint();
177                             }
178                             else if (i == TSEMI)
179                             {
180                                     return semiPrint();
181                             }
182                             else
183                             {
184                                     cbuf = fgetc(fp);
185                                     printf(" %s", tokenstr[i]);
186                                     return i;
187                             }
188                     }
189             }
190             if (cbuf == EOF)
191             {
192                     return EOFCODE;
193             }
194
```

```
195                 char dst[100];
196                 snprintf(dst, sizeof dst, "%c is undeclared.\n", cbuf);
197                 error(dst);
198                 return ERROR;
199         }
200 }
201
202 int isNumber(int c)
203 {
204         if (c >= '0' && c <= '9')
205         {
206                 return 1;
207         }
208         else
209         {
210                 return 0;
211         }
212 }
213
214 int isChar(int c)
215 {
216         if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
217         {
218                 return 1;
219         }
220         else
221         {
222                 return 0;
223         }
224 }
225
226 void UntilFun(int c)
227 {
228         char cin = cbuf;
229         cbuf = fgetc(fp);
230         while (cbuf != c)
231         {
232                 if (cbuf == EOF)
233                 {
234                         char mes[100];
235                         sprintf(mes, "%c is undeclared. %c is expected.\n", cin, c);
236                         error(mes);
237                         break;
238                 }
239                 cbuf = fgetc(fp);
240         }
241         cbuf = fgetc(fp);
242 }
243
```

```
244 void UntilComment(void)
245 {
246         while (1)
247         {
248                 cbuf = fgetc(fp);
249                 if (cbuf == '*')
250                 {
251                         cbuf = fgetc(fp);
252                         if (cbuf == '/')
253                         {
254                                 cbuf = fgetc(fp);
255                                 break;
256                         }
257                 }
258                 if (cbuf == EOF)
259                 {
260                         error("/* is undeclared.another */ is expected.\n");
261                         break;
262                 }
263         }
264 }
265
266 void UntilString(void)
267 {
268         char sttemp[MAXSTRSIZE];
269         int i = 0;
270         sttemp[0] = '\'';
271         while (i < MAXSTRSIZE)
272         {
273                 cbuf = fgetc(fp);
274                 i++;
275                 sttemp[i] = cbuf;
276                 if (cbuf == '\'')
277                 {
278                         cbuf = fgetc(fp);
279                         if (cbuf != '\'')
280                         {
281                                 int j = 0;
282                                 printf(" ");
283                                 while (j <= i)
284                                 { // print string
285                                         printf("%c", sttemp[j]);
286                                         j++;
287                                 }
288                                 break;
289                         }
290                         else
291                         {
292                                 i++;
```

```
293                             sttemp[i] = '\'';
294                     }
295             }
296             if (cbuf == EOF)
297             {
298                     error("\' is undeclared.another \' is expected.\n");
299                     break;
300             }
301     }
302 }
303
304 int get_linenum(void)
305 {
306     return num_line;
307 }
308 void end_scan(void)
309 {
310     fclose(fp);
311 }
312
313 void print_indent(void)
314 {
315     int i = 0;
316     printf("\n");
317     for (i = 1; i < num_indent; i++)
318     {
319         printf(" ");
320     }
321 }
322
323 void print_tab(void)
324 {
325     int i = 0;
326     for (i = 1; i < num_indent; i++)
327     {
328         printf(" ");
329     }
330 }
331
332 void init_string_atr(int count)
333 {
334     prestate = OTEHER;
335     while (count >= 0)
336     { // init string_attr
337         string_attr[count] = '\0';
338         count--;
339     }
340 }
341
```

```
342 int programPrint(int count)
343 { // print process when token is program
344         print_indent();
345         printf(" %s", string_attr);
346         init_string_atr(count);
347         num_indent += 1;
348         return TPROGRAM;
349 }
350
351 int beginPrint(int count)
352 { // print process when token is begin
353         if (prestate == THEN)
354         {
355                 num_then--;
356                 num_indent--;
357                 printf("\r");
358                 print_tab();
359                 printf(" %s", string_attr);
360                 num_indent += 1;
361                 print_indent();
362                 init_string_atr(count);
363                 return TBEGIN;
364         }
365         else if (prestate == ELSE)
366         {
367                 print_indent();
368                 printf(" %s", string_attr);
369                 num_indent += 1;
370                 print_indent();
371                 init_string_atr(count);
372                 return TBEGIN;
373         }
374         else if (prestate != SEMI)
375         {
376                 print_indent();
377                 printf(" %s", string_attr);
378                 num_indent += 1;
379                 print_indent();
380                 init_string_atr(count);
381                 return TBEGIN;
382         }
383         else
384         {
385                 printf(" %s", string_attr);
386                 num_indent += 1;
387                 print_indent();
388                 init_string_atr(count);
389                 return TBEGIN;
390         }
```

```
391 }
392
393 int endPrint(int count)
394 { // print process when token is end
395         if (prestate == SEMI)
396         {
397                 if (num_then > 0)
398                 {
399                         num_then -= 1;
400                         printf("\r");
401                         num_indent -= 2;
402                         print_tab();
403                         printf(" %s", string_attr);
404                         init_string_atr(count);
405                         return TEND;
406                 }
407                 printf("\r");
408                 num_indent -= 1;
409                 print_tab();
410                 printf(" %s", string_attr);
411                 init_string_atr(count);
412                 return TEND;
413         }
414         if (num_then > 0)
415         {
416                 num_then -= 1;
417                 printf("\r");
418                 num_indent -= 1;
419                 print_tab();
420                 printf(" %s", string_attr);
421                 init_string_atr(count);
422                 return TEND;
423         }
424         num_indent -= 1;
425         print_indent();
426         printf(" %s", string_attr);
427         init_string_atr(count);
428         return TEND;
429 }
430
431 int ifPrint(int count)
432 { // print process when token is if
433         if (num_then > 0)
434         {
435                 num_then -= 1;
436                 printf("\r");
437                 num_indent -= 1;
438                 print_tab();
439         }
```

```
440         printf(" %s", string_attr);
441         init_string_atr(count);
442         return TIF;
443 }
444
445 int elsePrint(int count)
446 { // print process when token is else
447         if (num_then > 0)
448         {
449                 num_then--;
450                 printf("\r");
451                 num_indent--;
452                 print_indent();
453                 printf(" %s", string_attr);
454                 init_string_atr(count);
455                 prestate = ELSE;
456                 return TELSE;
457         }
458         else if (prestate == SEMI)
459         {
460                 printf(" %s", string_attr);
461                 init_string_atr(count);
462                 prestate = ELSE;
463                 return TELSE;
464         }
465         else
466         {
467                 print_indent();
468                 printf(" %s", string_attr);
469                 init_string_atr(count);
470                 prestate = ELSE;
471                 return TELSE;
472         }
473 }
474
475 int thenPrint(int count)
476 { // print process when token is then
477         printf(" %s", string_attr);
478         init_string_atr(count);
479         num_indent += 1;
480         print_indent();
481         num_then += 1;
482         prestate = THEN;
483         return TTHEN;
484 }
485
486 int noteqPrint(void)
487 {
488         cbuf = fgetc(fp);
```

```
489         if (cbuf == '>')
490         {
491                 cbuf = fgetc(fp);
492                 printf("%s", tokenstr[TNOTEQ]);
493                 return TNOTEQ;
494         }
495         else if (cbuf == '=')
496         {
497                 cbuf = fgetc(fp);
498                 printf("%s", tokenstr[TLEEQ]);
499                 return TLEEQ;
500         }
501         else
502         {
503                 printf("%s", tokenstr[TLE]);
504                 return TLE;
505         }
506 }
507
508 int grPrint(void)
509 {
510         cbuf = fgetc(fp);
511         if (cbuf == '=')
512         {
513                 cbuf = fgetc(fp);
514                 printf("%s", tokenstr[TGREQ]);
515                 return TGREQ;
516         }
517         else
518         {
519                 printf("%s", tokenstr[TGR]);
520                 return TGR;
521         }
522 }
523
524 int assignPrint(void)
525 {
526         cbuf = fgetc(fp);
527         if (cbuf == '=')
528         {
529                 cbuf = fgetc(fp);
530                 printf(" %s", tokenstr[TASSIGN]);
531                 return TASSIGN;
532         }
533         else
534         {
535                 printf("%s", tokenstr[TCOLON]);
536                 return TCOLON;
537         }
```

```
538 }
539
540 int semiPrint(void)
541 {
542         prestate = SEMI;
543         cbuf = fgetc(fp);
544         printf("%s", tokenstr[TSEMI]);
545         print_indent();
546         return TSEMI;
547 }
```

ソースコード 71 ebnf-list.c

```c
#include "pprinter-list.h"
#define NOMAL 0
#define ERROR 1
int token;

void set_token(int t)
{
    token = t;
}

int parse_program()
{
    // program ::= "program" NAME ";" block "."
    if (token != TPROGRAM)
    {
        error("Keyword 'program' is not found\n");
        // return ERROR;
    }
    token = scan();
    if (token != TNAME)
    {
        error("Program name is not found\n");
    }
    token = scan();
    if (token != TSEMI)
    {
        error("';' is not found in parse_program\n");
    }
    token = scan();
    if (parse_block() == ERROR)
    {
        // error("Block is not found\n");
    }
    if (token != TDOT)
    {
        error("'.' is not found at the end of program\n");
    }
    printf("\n");
    token = scan();
    return NOMAL;
}

int parse_block()
{
    // block ::= {variable_declaration | procedure_declaration} compound_statement
    while ((token == TVAR) || (token == TPROCEDURE))
    {
```

```
48          if (token == TVAR)
49          {
50              if (parse_variable_declaration() == ERROR)
51              {
52                  // error("Variable declaration is not found in block\n");
53              }
54          }
55          else if (token == TPROCEDURE)
56          {
57              if (parse_subprogram_declaration() == ERROR)
58              {
59                  // error("Subprogram declaration is not found in block\n");
60              }
61          }
62      }
63      if (parse_compound_statement() == ERROR)
64      {
65          // error("Compound statement is not found in block\n");
66      }
67      return NOMAL;
68  }
69
70  int parse_variable_declaration()
71  {
72      // variable_declaration ::= "var" variable_names ":" type ";" {variable_names ":" type
            ";"}
73
74      // when call this function, token is "var"
75      // if(token != TVAR){
76      // error("Keyword 'var' is not found in variable_declaration\n");
77      // }
78      token = scan();
79      if (parse_variable_names() == ERROR)
80      {
81          // error("Variable name is not found in variable_declaration\n");
82      }
83      if (token != TCOLON)
84      {
85          error("':' is not found in variable_declaration1\n");
86      }
87      token = scan();
88      if (parse_type() == ERROR)
89      {
90          // error("Type is not found in variable_declaration\n");
91      }
92      if (token != TSEMI)
93      {
94          error("';' is not found in variable_declaration\n");
95      }
```

```
96      token = scan();
97      while (token == TNAME)
98      {
99          if (parse_variable_names() == ERROR)
100         {
101             // error("Variable name is not found in variable_declaration\n");
102         }
103         if (token != TCOLON)
104         {
105             error("':' is not found in variable_declaration2\n");
106         }
107         token = scan();
108         if (parse_type() == ERROR)
109         {
110             // error("Type is not found in variable_declaration\n");
111         }
112         if (token != TSEMI)
113         {
114             error("';' is not found in variable_declaration\n");
115         }
116         token = scan();
117     }
118     return NOMAL;
119 }
120
121 int parse_variable_names()
122 {
123     // variable_names ::= NAME {"," NAME}
124     if (token != TNAME)
125     {
126         error("Variable name is not found in variable_names1\n");
127     }
128     token = scan();
129     while (token == TCOMMA)
130     {
131         token = scan();
132         if (token != TNAME)
133         {
134             error("Variable name is not found in variable_names2\n");
135         }
136         token = scan();
137     }
138     return NOMAL;
139 }
140
141 int parse_type()
142 {
143     // type ::= standard_type | array_type
144     if (token == TINTEGER || token == TBOOLEAN || token == TCHAR)
```

```
145     {
146         if (parse_standard_type() == ERROR)
147         {
148             // error("Standard type is not found in type\n");
149         }
150     }
151     else if (token == TARRAY)
152     {
153         if (parse_array_type() == ERROR)
154         {
155             // error("Array type is not found in type\n");
156         }
157     }
158     else
159     {
160         error("Type is not found in type\n");
161     }
162     return NOMAL;
163 }
164 
165 int parse_standard_type()
166 {
167     // standard_type ::= "integer" | "boolean" | "char"
168     switch (token)
169     {
170     case TINTEGER:
171     case TBOOLEAN:
172     case TCHAR:
173         token = scan();
174         break;
175     default:
176         error("Standard type is not found in standard_type\n");
177     }
178     return NOMAL;
179 }
180 
181 int parse_array_type()
182 {
183     // array_type ::= "array" "[" NUMBER "]" "of" standard_type
184 
185     // when call this function, token is "array"
186     // if(token != TARRAY){
187     // error("Keyword 'array' is not found in array_type\n");
188     // }
189     token = scan();
190     if (token != TLSQPAREN)
191     {
192         error("'[' is not found in array_type\n");
193     }
```

```
194     token = scan();
195     if (token != TNUMBER)
196     {
197         error("Number is not found in array_type\n");
198     }
199     token = scan();
200     if (token != TRSQPAREN)
201     {
202         error("']' is not found in array_type\n");
203     }
204     token = scan();
205     if (token != TOF)
206     {
207         error("Keyword 'of' is not found in array_type\n");
208     }
209     token = scan();
210     if (parse_standard_type() == ERROR)
211     {
212         // error("Standard type is not found in array_type\n");
213     }
214     return NOMAL;
215 }
216 
217 int parse_subprogram_declaration()
218 {
219     // subprogram_declaration ::= "procedure" NAME [formal_parameters] ";" [
            variable_declaration] compound_statement ";"
220 
221     // when call this function, token is "procedure"
222     // if(token != TPROCEDURE){
223     // error("Keyword 'procedure' is not found in subprogram_declaration\n");
224     // }
225     token = scan();
226     if (token != TNAME)
227     {
228         error("Procedure_name is not found in subprogram_declaration\n");
229     }
230     token = scan();
231     if (token == TLPAREN)
232     { // it is ok if formal_parameters is not found
233         if (parse_formal_parameters() == ERROR)
234         {
235             // error("Formal parameters is not found in subprogram_declaration\n");
236         }
237     }
238     if (token != TSEMI)
239     {
240         error("';' is not found in subprogram_declaration\n");
241     }
```

```
242     token = scan();
243     if (token == TVAR)
244     { // it is ok if variable_declaration is not found
245         if (parse_variable_declaration() == ERROR)
246         {
247             // error("Variable declaration is not found in subprogram_declaration\n");
248         }
249     }
250     if (parse_compound_statement() == ERROR)
251     {
252         // error("Compound statement is not found in subprogram_declaration\n");
253     }
254     if (token != TSEMI)
255     {
256         error("';' is not found in subprogram_declaration\n");
257     }
258     token = scan();
259     return NOMAL;
260 }
261 
262 int parse_formal_parameters()
263 {
264     // formal_parameters ::= "(" variable_names ":" type {";" variable_names ":" type} ")"
265 
266     // when call this function, token is "("
267     // if(token != TLPAREN){
268     // error("'(' is not found in formal_parameters\n");
269     // }
270     token = scan();
271     if (parse_variable_names() == ERROR)
272     {
273         // error("Variable name is not found in in formal_parameters\n");
274     }
275     if (token != TCOLON)
276     {
277         error("':' is not found in formal_parameters\n");
278     }
279     token = scan();
280     if (parse_type() == ERROR)
281     {
282         // error("Type is not found in in formal_parameters\n");
283     }
284     while (token == TSEMI)
285     {
286         token = scan();
287         if (parse_variable_names() == ERROR)
288         {
289             // error("Variable name is not found in in formal_parameters\n");
290         }
```

```
291         if (token != TCOLON)
292         {
293             error("':' is not found in formal_parameters in in formal_parameters\n");
294         }
295         token = scan();
296         if (parse_standard_type() == ERROR)
297         {
298             // error("Standard type is not found in formal_parameters\n");
299         }
300     }
301     if (token != TRPAREN)
302     {
303         error("')' is not found in formal_parameters\n");
304     }
305     token = scan();
306     return NOMAL;
307 }
308 
309 int parse_compound_statement()
310 {
311     // compound_statement ::= "begin" statement {";" statement} "end"
312     if (token != TBEGIN)
313     {
314         error("Keyword 'begin' is not found\n");
315     }
316     token = scan();
317     if (parse_statement() == ERROR)
318     {
319         // error("Statement is not found in compound_statement1\n");
320     }
321     while (token == TSEMI)
322     {
323         token = scan();
324         if (parse_statement() == ERROR)
325         {
326             // error("Statement is not found in compound_statement2\n");
327         }
328     }
329     if (token != TEND)
330     {
331         error("Keyword 'end' is not found in compound_statement\n");
332     }
333     token = scan();
334     return NOMAL;
335 }
336 
337 int parse_statement()
338 {
339     /*
```

```
340         statement ::= assignment_statement | condition_statement | iteration_statement |
341                       exit_statement | call_statement | return_statement | input_statement|
342                       output_statement | compound_statement | empty_statement
343     */
344     switch (token)
345     {
346     case TNAME:
347         if (parse_assignment_statement() == ERROR)
348         {
349             // error("Assignment statement is not found in parse_statement\n");
350         }
351         break;
352     case TIF:
353         if (parse_condition_statement() == ERROR)
354         {
355             // error("Condition statement is not found in parse_statement\n");
356         }
357         break;
358     case TWHILE:
359         if (parse_iteration_statement() == ERROR)
360         {
361             // error("Iteration statement is not found in parse_statement\n");
362         }
363         break;
364     case TBREAK:
365         if (parse_exit_statement() == ERROR)
366         {
367             // error("Exit statement is not found in parse_statement\n");
368         }
369         break;
370     case TCALL:
371         if (parse_call_statement() == ERROR)
372         {
373             // error("Call statement is not found in parse_statement\n");
374         }
375         break;
376     case TRETURN:
377         if (parse_return_statement() == ERROR)
378         {
379             // error("Return statement is not found in parse_statement\n");
380         }
381         break;
382     case TREAD:
383     case TREADLN:
384         if (parse_input_statement() == ERROR)
385         {
386             // error("Input statement is not found in parse_statement\n");
387         }
388         break;
```

```
389     case TWRITE:
390     case TWRITELN:
391         if (parse_output_statement() == ERROR)
392         {
393             // error("Output statement is not found in parse_statement\n");
394         }
395         break;
396     case TBEGIN:
397         if (parse_compound_statement() == ERROR)
398         {
399             // error("Compound statement is not found in parse_statement\n");
400         }
401         break;
402     default:
403         parse_empty_statement();
404     }
405     return NOMAL;
406 }
407 
408 int parse_condition_statement()
409 {
410     // condition_statement ::= "if" expression "then" statement ["else" statement]
411 
412     // when call this function, token is "if"
413     // if(token != TIF){
414     // error("Keyword 'if' is not found in condition_statement\n");
415     // }
416     token = scan();
417     if (parse_expression() == ERROR)
418     {
419         // error("Expression is not found in condition_statement\n");
420     }
421     if (token != TTHEN)
422     {
423         error("Keyword 'then' is not found in condition_statement\n");
424     }
425     token = scan();
426     if (parse_statement() == ERROR)
427     {
428         // error("Statement is not found in condition_statement\n");
429     }
430     if (token == TELSE)
431     { // it is ok if there is no else statement
432         token = scan();
433         if (token != TIF)
434         {
435             print_indent();
436         }
437         if (parse_statement() == ERROR)
```

```
438         {
439             // error("Statement is not found in condition_statement\n");
440         }
441     }
442     return NOMAL;
443 }
444 
445 int parse_iteration_statement()
446 {
447     // iteration_statement ::= "while" expression "do" statement
448 
449     // when call this function, token is "while"
450     // if(token != TWHILE){
451     // error("Keyword 'while' is not found in iteration_statement\n");
452     // }
453     token = scan();
454     if (parse_expression() == ERROR)
455     {
456         // error("Expression is not found in iteration_statement\n");
457     }
458     if (token != TDO)
459     {
460         error("Keyword 'do' is not found in iteration_statement\n");
461     }
462     token = scan();
463     if (parse_statement() == ERROR)
464     {
465         // error("Statement is not found in iteration_statement\n");
466     }
467     return NOMAL;
468 }
469 
470 int parse_exit_statement()
471 {
472     // exit_statement ::= "break"
473 
474     // when call this function, token is "break"
475     // if(token != TBREAK){
476     // error("Keyword 'break' is not found in exit_statement\n");
477     // }
478     token = scan();
479     return NOMAL;
480 }
481 
482 int parse_call_statement()
483 {
484     // call_statement ::= "call" NAME ["(" expressions ")"]
485 
486     // when call this function, token is "call"
```

```
487     // if(token != TCALL){
488     // error("Keyword 'call' is not found in call_statement\n");
489     // }
490     token = scan();
491     if (token != TNAME)
492     {
493         error("Procedure name is not found in call_statement\n");
494     }
495     token = scan();
496     if (token == TLPAREN)
497     { // it is ok if there is no expression
498         token = scan();
499         if (parse_expressions() == ERROR)
500         {
501             // error("Expressions are not found in call_statement\n");
502         }
503         if (token != TRPAREN)
504         {
505             error("')' is not found in call_statement\n");
506         }
507         token = scan();
508     }
509     return NOMAL;
510 }
511
512 int parse_expressions()
513 {
514     // expressions ::= expression {"," expression}
515     if (parse_expression() == ERROR)
516     {
517         // error("Expression is not found in expressions\n");
518     }
519     while (token == TCOMMA)
520     {
521         token = scan();
522         if (parse_expression() == ERROR)
523         {
524             // error("Expression is not found in expressions\n");
525         }
526     }
527     return NOMAL;
528 }
529
530 int parse_return_statement()
531 {
532     // return_statement ::= "return"
533
534     // when call this function, token is "return"
535     // if(token != TRETURN){
```

```
536     // error("Keyword 'return' is not found in return_statement\n");
537     // }
538     token = scan();
539     // return NOMAL;
540 }
541 
542 int parse_assignment_statement()
543 {
544     // assignment_statement ::= variable ":=" expression
545     if (parse_variable() == ERROR)
546     {
547         // error("Variable name is not found in return_statement\n");
548     }
549     if (token != TASSIGN)
550     {
551         error("':=' is not found in return_statement\n");
552     }
553     token = scan();
554     if (parse_expression() == ERROR)
555     {
556         // error("Expression is not found in return_statement\n");
557     }
558     return NOMAL;
559 }
560 
561 int parse_variable()
562 {
563     // variable = NAME ["[" expression "]"]
564     if (token != TNAME)
565     {
566         error("Variable name is not found in variable\n");
567     }
568     token = scan();
569     if (token == TLSQPAREN)
570     { // it is ok if there is no [ expression ]
571         token = scan();
572         if (parse_expression() == ERROR)
573         {
574             // error("Expression is not found in variable\n");
575         }
576         if (token != TRSQPAREN)
577         {
578             error("']' is not found in variable\n");
579         }
580         token = scan();
581     }
582     return NOMAL;
583 }
584 
```

```
585 int parse_expression()
586 {
587     // expression ::= simple_expression {relational_operator simple_expression}
588     if (parse_simple_expression() == ERROR)
589     {
590         // error("Simple expression is not found in expression\n");
591     }
592     if (token == TEQUAL || token == TNOTEQ || token == TLE || token == TLEEQ || token ==
          TGR || token == TGREQ)
593     {
594         token = scan();
595         if (parse_simple_expression() == ERROR)
596         {
597             // error("Simple expression is not found in expression\n");
598         }
599     }
600
601     return NOMAL;
602 }
603
604 int parse_simple_expression()
605 {
606     // simple_expression ::= ["+"|"-"] term {adding_operator term}
607     if (token == TPLUS || token == TMINUS)
608     { // it is ok if there is no sign
609         token = scan();
610     }
611     if (parse_term() == ERROR)
612     {
613         // error("Term is not found in simple_expression\n");
614     }
615     if (token == TPLUS || token == TMINUS || token == TOR)
616     {
617         while (token == TPLUS || token == TMINUS || token == TOR)
618         {
619             token = scan();
620             if (parse_term() == ERROR)
621             {
622                 // error("Term is not found in simple_expression\n");
623             }
624         }
625     }
626     return NOMAL;
627 }
628
629 int parse_term()
630 {
631     // term ::= factor {multiplying_operator factor}
632     if (parse_factor() == ERROR)
```

```
633     {
634         // error("Factor is not found in term\n");
635         // return(ERROR);
636     }
637 
638     while (token == TSTAR || token == TAND || token == TDIV)
639     {
640         token = scan();
641         if (parse_factor() == ERROR)
642         {
643             // error("Factor is not found in term\n");
644         }
645     }
646     return NOMAL;
647 }
648 
649 int parse_factor()
650 {
651     // factor ::= variable | constant | "(" expression ")" | "not" factor | standard_type
            "(" expression ")"
652     if (token == TNAME)
653     {
654         if (parse_variable() == ERROR)
655         {
656             // error("Variable is not found in factor\n");
657         }
658     }
659     else if (token == TNUMBER || token == TSTRING || token == TFALSE || token == TTRUE)
660     {
661         token = scan();
662     }
663     else if (token == TLPAREN)
664     {
665         token = scan();
666         if (parse_expression() == ERROR)
667         {
668             // error("Expression is not found in factor\n");
669         }
670         if (token != TRPAREN)
671         {
672             error("')' is not found in factor\n");
673         }
674         token = scan();
675     }
676     else if (token == TNOT)
677     {
678         token = scan();
679         if (parse_factor() == ERROR)
680         {
```

```
681             // error("Factor is not found in factor\n");
682         }
683     }
684     else if (token == TINTEGER || token == TBOOLEAN || token == TCHAR)
685     {
686         token = scan();
687         if (token != TLPAREN)
688         {
689             error("'(' is not found in factor\n");
690         }
691         token = scan();
692         if (parse_expression() == ERROR)
693         {
694             // error("Expression is not found in factor\n");
695         }
696         if (token != TRPAREN)
697         {
698             error("')' is not found in factor\n");
699         }
700         token = scan();
701     }
702     else
703     {
704         error("Factor is not found in factor\n");
705     }
706     return NOMAL;
707 }
708 
709 /***
710 
711 // instead of parse_standard_type, NUMBER, STRING, FALSE, TRUE are used
712 int parse_constant(){
713     // constant ::= "NUMBER" | "false" | "true" | "STRING"
714     if(token == TNUMBER || token == TSTRING || token == TFALSE || token == TTRUE){
715         token = scan();
716         return NOMAL;
717     }
718     error("Constant is not found in constant\n");
719 }
720 
721 // instead of parse_standard_type, INTEGER, BOOLEAN, CHAR are used
722 int parse_multiplicative_operator(){
723     // multiplicative_operator ::= "*" | "div" | "and"
724     if(token == TSTAR || token == TDIV || token == TAND){
725         token = scan();
726         return NOMAL;
727     }
728     error("Multiplicative operator is not found in multiplicative_operator\n");
729 }
```

```
730
731 // instead of parse_standard_type, TPLUS, TMINUS, TOR are used
732 int parse_additive_operator(){
733     // adding_operator ::= "+" | "-" | "or"
734     if(token == TPLUS || token == TMINUS || token == TOR){
735         token = scan();
736         return NOMAL;
737     }
738     error("Additive operator is not found in additive_operator\n");
739     return ERROR;
740 }
741
742 // instead of parse_standard_type, TEQUAL, TNOTEQ, TLE, TLEEQ, TGR, TGREQ are used
743 int parse_relational_operator(){
744     // relational_operator ::= "=" | "<>" | "<" | "<=" | ">" | ">="
745     if(token == TEQUAL || token == TNOTEQ || token == TLE || token == TLEEQ || token == TGR
          || token == TGREQ){
746         token = scan();
747         return NOMAL;
748     }
749     error("Relational operator is not found in relational_operator\n");
750     return ERROR;
751 }
752
753 ***/
754
755 int parse_input_statement()
756 {
757     // input_statement ::= ("read" | "readln") ["(" variable {"," variable} ")"]
758
759     // when call this function, token is already "read" or "readln"
760     // if(token != TREAD && token != TREADLN){
761     // error("Keyword 'read' or 'readln' is not found in input_statement\n");
762     // }
763     token = scan();
764     if (token == TLPAREN)
765     { // it is ok if there is no variable
766         token = scan();
767         if (parse_variable() == ERROR)
768         {
769             // error("Variable is not found in input_statement\n");
770         }
771         while (token == TCOMMA)
772         {
773             token = scan();
774             if (parse_variable() == ERROR)
775             {
776                 // error("Variable is not found in input_statement\n");
777             }
```

```
778         }
779         if (token != TRPAREN)
780         {
781             error("')' is not found in input_statement\n");
782         }
783         token = scan();
784     }
785     return NOMAL;
786 }
787 
788 int parse_output_statement()
789 {
790     // output_statement ::= ("write" | "writeln") ["(" output_format {"," output_format}
        ")"]
791 
792     // when call this function, token is TWRITE or TWRITELN
793     // if(token != TWRITE && token != TWRITELN){
794     // error("Keyword 'write' or 'writeln' is not found in output_statement\n");
795     // }
796     token = scan();
797     if (token == TLPAREN)
798     { // it is ok if there is no output_format
799         token = scan();
800         if (parse_output_format() == ERROR)
801         {
802             // error("Output format is not found in output_statement\n");
803         }
804         while (token == TCOMMA)
805         {
806             token = scan();
807             if (parse_output_format() == ERROR)
808             {
809                 // error("Output format is not found in output_statement\n");
810             }
811         }
812         if (token != TRPAREN)
813         {
814             error("')' is not found in output_statement\n");
815         }
816         token = scan();
817     }
818     return NOMAL;
819 }
820 
821 int parse_output_format()
822 {
823     // output_format ::= expression [":" "NUMBER"] | "STRING"
824     if (token == TSTRING)
825     {
```

```
826         token = scan();
827     }
828     else
829     {
830         if (parse_expression() == ERROR)
831         {
832             // error("Expression is not found in output_format\n");
833         }
834         if (token == TCOLON)
835         { // it is ok if there is no number
836             token = scan();
837             if (token != TNUMBER)
838             {
839                 error("Number is not found in output_format\n");
840             }
841             token = scan();
842         }
843     }
844     return NOMAL;
845 }
846 
847 int parse_empty_statement()
848 {
849     // empty_statement ::= 竜
850     return NOMAL;
851 }
```

ソースコード 72　comand.sh

```
#!/bin/bash
echo "rm *.gcda *.gcno *.gcov"
rm *.gcda *.gcno *.gcov
echo "gcc --coverage -o tc id-list.c scan-list.c pprinter-list.c token-list.h ebnf-list.c"
gcc --coverage -o tc id-list.c scan-list.c pprinter-list.c pprinter-list.h ebnf-list.c
echo "./tc sample2a.mpl"
./tc sample2a.mpl
echo "./tc sample02a.mpl"
./tc sample02a.mpl
echo "./tc sample21.mpl"
./tc sample21.mpl
echo "./tc sample021.mpl"
./tc sample021.mpl
echo "./tc sample022.mpl"
./tc sample022.mpl
echo "./tc sample22.mpl"
./tc sample22.mpl
echo "./tc sample23.mpl"
./tc sample23.mpl
echo "./tc sample023.mpl"
./tc sample023.mpl
echo "./tc sample024.mpl"
./tc sample024.mpl
echo "./tc sample24.mpl"
./tc sample24.mpl
echo "./tc sample25.mpl"
./tc sample25.mpl
echo "./tc sample025.mpl"
./tc sample025.mpl
echo "./tc sample025t.mpl"
./tc sample025t.mpl
echo "./tc sample25t.mpl"
./tc sample25t.mpl
echo "./tc sample26.mpl"
./tc sample26.mpl
echo "./tc sample026.mpl"
./tc sample026.mpl
echo "./tc sample026t.mpl"
```

```
./tc sample026t.mpl
echo "./tc sample27.mpl"
./tc sample27.mpl
echo "./tc sample28p.mpl"
./tc sample28p.mpl
echo "./tc sample028p.mpl"
./tc sample028p.mpl
echo "./tc sample028q.mpl"
./tc sample028q.mpl
echo "./tc sample028r.mpl"
./tc sample028r.mpl
echo "./tc sample29p.mpl"
./tc sample29p.mpl
echo "./tc sample029p.mpl"
./tc sample029p.mpl
echo "./tc sample029q.mpl"
./tc sample029q.mpl
echo "./tc sample029r.mpl"
./tc sample029r.mpl
echo "./tc sample029s.mpl"
./tc sample029s.mpl
echo "./tc sample029t.mpl"
./tc sample029t.mpl
echo "test1.mpl"
./tc test1.mpl
echo "test2.mpl"
./tc test2.mpl
echo "test3.mpl"
./tc test3.mpl
echo "test4.mpl"
./tc test4.mpl
echo "test5.mpl"
./tc test5.mpl
echo "test6.mpl"
./tc test6.mpl
echo "test7.mpl"
./tc test7.mpl
echo "test8.mpl"
./tc test8.mpl
echo "test9.mpl"
```

```
./tc test9.mpl
echo "test10.mpl"
./tc test10.mpl
echo "test11.mpl"
./tc test11.mpl
echo "test12.mpl"
./tc test12.mpl
echo "test13.mpl"
./tc test13.mpl
echo "test14.mpl"
./tc test14.mpl
echo "test15.mpl"
./tc test15.mpl
echo "test16.mpl"
./tc test16.mpl
echo "tb1.mpl"
./tc tb1.mpl
echo "tb2.mpl"
./tc tb2.mpl
echo "tb3.mpl"
./tc tb3.mpl
echo "tb4.mpl"
./tc tb4.mpl
echo "tb5.mpl"
./tc tb5.mpl
echo "tb6.mpl"
./tc tb6.mpl
echo "tb7.mpl"
./tc tb7.mpl
echo "tb8.mpl"
./tc tb8.mpl
echo "tb9.mpl"
./tc tb9.mpl
echo "tb10.mpl"
./tc tb10.mpl
echo "tb11.mpl"
./tc tb11.mpl
echo "tb12.mpl"
./tc tb12.mpl
echo "tb13.mpl"
```

```
./tc tb13.mpl
echo "tb14.mpl"
./tc tb14.mpl
echo "tb15.mpl"
./tc tb15.mpl
echo "tb16.mpl"
./tc tb16.mpl
echo "tb17.mpl"
./tc tb17.mpl
echo "tb18.mpl"
./tc tb18.mpl
echo "tb19.mpl"
./tc tb19.mpl
echo "tb20.mpl"
./tc tb20.mpl
echo "tb21.mpl"
./tc tb21.mpl
echo "tb22.mpl"
./tc tb22.mpl
echo "tb23.mpl"
./tc tb23.mpl
echo "tt1.mpl"
./tc tt1.mpl
echo "tt2.mpl"
./tc tt2.mpl
echo "tt3.mpl"
./tc tt3.mpl
echo "tt4.mpl"
./tc tt4.mpl
echo "tt5.mpl"
./tc tt5.mpl
echo "tt6.mpl"
./tc tt6.mpl
echo "tt7.mpl"
./tc tt7.mpl
echo "ta1.mpl"
./tc ta1.mpl
echo "ta2.mpl"
./tc ta2.mpl
echo "ta3.mpl"
```

```
./tc ta3.mpl
echo "ta4.mpl"
./tc ta4.mpl
echo "ta5.mpl"
./tc ta5.mpl
echo "ta6.mpl"
./tc ta6.mpl
echo "ta7.mpl"
./tc ta7.mpl
echo "ta8.mpl"
./tc ta8.mpl
echo "ta9.mpl"
./tc ta9.mpl
echo "ta10.mpl"
./tc ta10.mpl
echo "ta11.mpl"
./tc ta11.mpl
echo "ta12.mpl"
./tc ta12.mpl
echo "gcov -b ebnf-list.gcda"
gcov −b ebnf−list.gcda
```

ソースコード 73 test1.mpl

```
1 aaa
```

ソースコード 74 test2.mpl

```
1 program
```

ソースコード 75 test3.mpl

```
1 program aa
```

ソースコード 76 test4.mpl

```
1 program aa; var
```

ソースコード 77 test05.mpl

```
1 program aa; var 1
```

ソースコード 78 test6.mpl

```
1 program aa; var a
```

ソースコード 79 test7.mpl

```
1 program aa; var a:
```

ソースコード 80 test8.mpl

```
1 program aa; var a: integer
```

ソースコード 81 test9.mpl

```
1 program aa; var a: integer; a
```

ソースコード 82 test10.mpl

```
1 program aa; var a: integer; a:
```

ソースコード 83 test11.mpl

```
1 program aa; var a: integer; a: integer
```

ソースコード 84 test12.mpl

```
1 program aa; var a: integer; a: integer;
```

ソースコード 85 test13.mpl

```
1 program aa; var a: integer; a: integer;
2 begin
```

ソースコード 86 test14.mpl

```
1 program aa; var a: integer; a: integer;
2 begin end
```

ソースコード 87 test15.mpl

```
1 program aa; var a: integer; a: integer;
2 begin end.
```

ソースコード 88 tm1.mpl

```
1  program
2  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
3  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
4  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
5  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
6  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
7  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
8  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
9  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
10 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
11 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
12 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
14 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
15 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
16 ;var a: integer; a: integer;
17 begin end.
```

ソースコード 89 tm2.mpl

```
1  program
2  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
3  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
4  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
5  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
6  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
7  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
8  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
9  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
10 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
11 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
12 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
14 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
15 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
16 ;var a: integer; a: integer;
17 begin end.
```

ソースコード 90　tm3.mpl

```
1  program
2  a
3  ;var b: integer; a: integer;
4  begin b := 2147483647end.
```

ソースコード 91　tm4.mpl

```
1  program
2  a
3  ;var b: integer; a: integer;
4  begin b := 2147483648 end.
```

他のテストデータ https://github.com/tyukei/Compiler/tree/main/Exe2

7　参考文献

参考文献

[1] https://www.techscore.com

[2] gcov カバレッジ説明 https://superactionshootinggame4.hatenablog.com/entry/2020/03/11/145907

[3] ソフトウェア開発見積りの基本的な考え方 https://www.ipa.go.jp/files/000005394.pdf