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

■cross-main.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。cros-main.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。

■id-list.c
記号表の初期化、挿入、表示を行うモジュール。出現した行や、名前、配列等を線形リストにて記録する。

■cross-main.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];
```

■idroot:単方向リスト

拡張機能としてトークン NAME が返されたとき名前の実体もカウントする実装をおこなった。実体はトークンと異なり、事前にどの種類の者があるか分からない。そこで、情報を後からつなげていくことのできる単方向リストを用いた。単方向リストとは各要素が自分の「次」の要素へのリンクを持ち、先頭側から末尾側へのみたどっていくデータ構造である。[1]。今回は、struct ID を用いて単方向リストを実現させた。value は NAME の実体の name とその実体の数の count、型 ttype, 定義行 defline, 参照行 refp, パラメータの型 paramp を持つ。そして nextp が次の格納場所 (アドレス) を保持する。　参照の行は何個参照されたかで変わるので、これも線形リストを用いて実現させた。参照行 reflinenum と次の格納場所をあらわす nextlinep を持つ。パラメータも同様に何個パラメータを持つか分からないので、線形リストで実現をさせた。パラメータの型 ttpe と次の格納場所を表す nextp を持つ。

ソースコード 2　構造体 ID in id-list.c

```
1  struct LINE
2  {
3      int reflinenum;
4      struct LINE *nextlinep;
5  };
6  struct PARAMETER
7  {
8      int ttype;
9      struct PARAMETER *nextp;
10 };
11 struct ID
12 {
13     char *name;
14     char *procname;
15     int count;
16     int ttype;
17     int defline;
18     struct LINE *refp;
19     struct PARAMETER *paramp;
20     struct ID *nextp;
21 };
```

■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 3　各変数 in cross-main.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 4　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　search_idtab(char *np) 関数

■機能　変数 np と同じ名前がすでに記号表に登録されているか検索する。

■引数　検索したい文字列。

■戻り値　すでに登録されている場合は、登録されているリストのポインタを返し、ない場合は NULL を返す。

ソースコード 5　search_idtab(char *np)

```
1  struct ID *search_idtab(char *np)
2  { /* search the name pointed by np */
3        struct ID *p;
4        if (strcmp(subprogramname, "") == 0)
5        {
6              for (p = head; p != NULL; p = p->nextp)
7              {
8                    if ((strcmp(p->procname, "") == 0) && (strcmp(p->name, np) == 0))
9                    {
10                         return p;
11                   }
12             }
13       }
14
15       else
16       {
17             for (p = head; p != NULL; p = p->nextp)
18             {
19                   if (strcmp(p->name, np) == 0 && strcmp(p->procname, subprogramname)
                         == 0)
20                   {
21                         return p;
22                   }
23             }
24       }
25       return NULL;
26 }
```

3.4.2　init_string_atr 関数

■機能　記号表に名前を登録し、カウントアップする。

■引数　登録したい文字列。

■戻り値　なし。

ソースコード 6　id_countup()

```
1  void id_countup(char *np)
2  { /* Register and count up the name pointed by np */
3        struct ID *p;
4        char *cp, *cp2;
5        if ((p = search_idtab(np)) != NULL)
6        {
7              p->count++;
```

```
8         }
9         else
10        {
11                if ((p = (struct ID *)malloc(sizeof(struct ID))) == NULL)
12                {
13                        error("can not malloc in id_countup\n");
14                }
15                if ((cp = (char *)malloc(strlen(np) + 1)) == NULL)
16                {
17                        error("can not malloc-2 in id_countup\n");
18                }
19                if ((cp2 = (char *)malloc(strlen(subprogramname) + 1)) == NULL)
20                {
21                        error("can not malloc-3 in id_countup\n");
22                }
23                strcpy(cp, np);
24                strcpy(cp2, subprogramname);
25                p->name = cp;
26                p->procname = cp2;
27                p->count = 1;
28                p->nextp = NULL;
29                p->refp = NULL;
30                p->paramp = NULL;
31                p->ttype = 0;
32                if (head == NULL)
33                {
34                        head = p;
35                        tail = p;
36                }
37                else
38                {
39                        tail->nextp = p;
40                        tail = p;
41                }
42        }
43  }
```

3.4.3　setSubprogramName 関数

■機能　副プログラムの名前を登録する関数

■引数　副プログラムの名前。

■戻り値　なし。

ソースコード 7　setSubprogramName()

```
1   void setSubprogramName(char *np)
2   {
3         char *cp;
```

```
4       if ((cp = (char *)malloc(strlen(np) + 1)) == NULL)
5       {
6               printf("can not malloc-2 in id_countup\n");
7               return;
8       }
9       strcpy(cp, np);
10      strcpy(subprogramname, np);
11  }
```

3.4.4　setSubprogramName 関数

■機能　定義行列を登録する関数

■引数　定義行列。

■戻り値　なし。

ソースコード 8　setDefLine()

```
1   void setDefLine(int line)
2   {
3       tail->defline = line;
4   }
```

3.4.5　setSubprogramName 関数

■機能　参照行列を登録する関数

■引数　参照行列。

■戻り値　なし。

ソースコード 9　setRefLine()

```
1   void setRefLine(char *np, int line)
2   {
3       struct ID *p;
4       struct LINE *rp;
5
6       if ((p = search_idtab(np)) != NULL)
7       {
8               rp = p->refp;
9               while (rp != NULL)
10              {
11                      rp = rp->nextlinep;
12              }
13              if ((rp = (struct LINE *)malloc(sizeof(struct LINE))) == NULL)
14              {
15                      error("can not malloc in id_countup\n");
16              }
17              rp->reflinenum = line;
```

```
18                      rp->nextlinep = NULL;
19                      if (p->refp == NULL)
20                      {
21                              p->refp = rp;
22                      }
23                      else
24                      {
25                              p->refp->nextlinep = rp;
26                      }
27              }
28      }
```

3.4.6　setIdtype

■機能　記号表に型を登録する関数

■引数　登録したい型のトークン。

■戻り値　なし。

ソースコード 10　setIdtype()

```
1  void setIdtype(int ttype)
2  {
3          char type[MAXSTRSIZE];
4          if (ttype == TINTEGER)
5          {
6                  strcpy(type, "integer");
7          }
8          else if (ttype == TBOOLEAN)
9          {
10                 strcpy(type, "boolean");
11         }
12         else if (ttype == TCHAR)
13         {
14                 strcpy(type, "char");
15         }
16         else if (ttype == TPROCEDURE)
17         {
18                 printf("procedure: %s\n", tail->name);
19                 strcpy(type, "procedure");
20                 struct PARAMETER *pp;
21                 if ((pp = (struct PARAMETER *)malloc(sizeof(struct PARAMETER))) == NULL)
22                 {
23                         error("can not malloc in id_countup\n");
24                 }
25                 pp->ttype = ttype;
26                 tail->paramp = pp;
27                 strcpy(tail->procname, "");
28         }
```

```
29          // printf("%s ", type);
30          if (isParameter)
31          {
32                  struct PARAMETER *pp;
33                  if ((pp = (struct PARAMETER *)malloc(sizeof(struct PARAMETER))) == NULL)
34                  {
35                          error("can not malloc in id_countup\n");
36                  }
37                  while (tail->paramp != NULL)
38                  {
39                          if (tail->paramp->ttype == 0)
40                          {
41                                  tail->paramp->ttype = ttype;
42                          }
43                  }
44                  isParameter = false;
45          }
46          else
47          {
48                  struct ID *p;
49                  if ((p = (struct ID *)malloc(sizeof(struct ID))) == NULL)
50                  {
51                          error("can not malloc in id_countup\n");
52                  }
53                  for (p = head; p != NULL; p = p->nextp)
54                  {
55                          if (p->ttype == 0)
56                          {
57                                  p->ttype = ttype;
58                          }
59                  }
60                  tail->ttype = ttype;
61          }
62  }
```

3.5　各関数の外部仕様 (課題 2)

3.5.1　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 11　init_string_atr()

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

3.5.2　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 12　set_token()

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

3.5.3　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 13　parse_program()

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

3.5.4　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 14　parse_block()

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

3.5.5　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 15　parse_variable_declaration()

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

3.5.6　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 16　parse_variable_names()

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

3.5.7　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 17　parse_type()

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

3.5.8　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 18　parse_standard_type()

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

3.5.9　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 19　parse_array_type()

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

3.5.10　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 20　parse_subprogram_declaration()

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

3.5.11　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 21　parse_formal_parameters()

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

3.5.12　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 22　parse_compound_statement()

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

3.5.13　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 23　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     */
```

3.5.14　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 24　parse_condition_statement()

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

3.5.15　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 25　parse_iteration_statement()

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

3.5.16　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 26　parse_exit_statement()

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

3.5.17　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 27　parse_call_statement()

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

3.5.18　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 28　parse_expressions()

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

3.5.19　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 29　parse_return_statement()

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

3.5.20　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 30　parse_assignment_statement()

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

3.5.21　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 31　parse_variable()

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

3.5.22　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 32　parse_expression()

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

3.5.23　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 33　parse_simple_expression()

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

3.5.24　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 34　parse_term()

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

3.5.25　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 35　parse_factor()

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

3.5.26　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 36　parse_constant()

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

3.5.27　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 37　parse_multiplicative_operator()

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

3.5.28　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 38　parse_input_statement()

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

3.5.29　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 39　parse_output_statement()

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

3.5.30　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 40　parse_output_format()

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

3.5.31　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 41　parse_empty_statement()

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

3.5.32　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 42　programPrint

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

3.5.33　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 43　beginPrint

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

3.5.34　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 44　ifPrint

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

3.5.35　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 45　elsePrint

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

3.5.36　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 46　thenPrint

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

3.5.37　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 47　noteqPrint

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

3.5.38　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 48　grPrint

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

3.5.39　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 49　assignPrint

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

3.5.40　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 50　semiPrint

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

3.5.41　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 51　endPrint

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

3.6　各関数の外部仕様 (課題 1)

3.6.1　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 52　init_scan()

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

3.6.2　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 53　scan()

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

3.6.3　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 54　get_linenum()

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

3.6.4　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 55　end_scan()

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

3.6.5　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 56　isChar()

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

3.6.6　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 57　UntilFun()

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

3.6.7　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 58　UntilComment()

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

3.6.8　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 59　UntilString()

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

3.6.9　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 60　init_idtab()

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

3.6.10　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 61　id_countup()

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

3.6.11　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 62　print_idtab()

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

3.6.12　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 63　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

- sample31p.mpl

- sample032p.mpl
- sample033p.mpl
- sample034.mpl
- sample35.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 64　演 sampl31p.mpl の実行例

```
$ gcc −o program pprinter−list.c pprinter−list.h ebnf−list.c scan−list.c
$ ./program sample31p.mpl
```

結果は以下のようになった。

ソースコード 65　result

```
Name Type Def. Ref.
a integer 2 22, 22,
p procedure(char) 3 21,
b char 8 20, 21,
q procedure(integer) 9 22, 22,
a:q boolean 10 15, 16,
q:q integer 14 15,
c integer 20
```

sample31p.mpl は 66 である。

ソースコード 66　sample1p.mpl

```
program sample31p;
var a : integer;
procedure p(a : char);
begin
    writeln('proc of p');
    a := 'a'
end;
var b : char;
procedure q(b:integer);
  var a : boolean;
      q : integer;
begin
    writeln('proc of q');
    readln(q);
    a := b = q;
    if a then writeln('true') else writeln('false')
end;
var c : integer;
begin
    a := 1;  b := 'b';
    call p(b);
    call q(a);call q(2*a+1)
end.
```

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 67　a を 1024 個書いたときの結果

```
1  program aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
2        .....
3      aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaatm1.mpl;
4       var a: integer;
5       a: integer;
6       begin
7
8       end .
```

ソースコード 68　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 69　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 70　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 ファイルを用いてホワイトボックステストを実行した。以下のようなコマンドを実行した。

ソースコード 71　comand.sh

```
#!/bin/bash
echo "rm *.gcda *.gcno *.gcov"
rm *.gcda *.gcno *.gcov
echo "gcc --coverage -o tc scan-list.c cross-main.c cross-main.h ebnf-list.c id-list2.c"
gcc --coverage -o tc scan-list.c cross-main.c cross-main.h ebnf-list.c id-list2.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 "tm1.mpl"
./tc tm1.mpl
echo "tm2.mpl"
./tc tm2.mpl
echo "tm3.mpl"
./tc tm3.mpl
echo "tm4.mpl"
./tc tm4.mpl
echo "sample31p.mpl"
./tc sample31p.mpl
echo "sample032p.mpl"
./tc sample032p.mpl
echo "sample33p.mpl"
./tc sample33p.mpl
echo "sample34.mpl"
./tc sample34.mpl
echo "sample35.mpl"
./tc sample35.mpl
echo "gcov -b ebnf-list.gcda"
gcov −b id−list2.gcda
```

ソースコード 72　bash を用いた comand の実行例

```
$ bash comand.sh
```

ソースコード 73　comand.sh の結果 1

```
rm *.gcda *.gcno *.gcov
gcc --coverage -o tc scan-list.c cross-main.c cross-main.h ebnf-list.c id-list2.c
./tc sample2a.mpl
Name Type Def. Ref.
n 3 5, 35,
count integer 3 5, 37,
./tc sample02a.mpl
Name Type Def. Ref.
./tc sample21.mpl
Name Type Def. Ref.
n integer 2 2,
./tc sample021.mpl

ERROR(2): ';' is not found in parse_program
Check grammar of 2 line in sample021.mpl

./tc sample022.mpl

ERROR(2): ':' is not found in variable_declaration1
Check grammar of 2 line in sample022.mpl

./tc sample22.mpl
Name Type Def. Ref.
x 3 9,
y boolean 3 10,
./tc sample23.mpl
Name Type Def. Ref.
x 2 4,
y char 2 4,
./tc sample023.mpl

ERROR(2): Keyword 'begin' is not found
Check grammar of 2 line in sample023.mpl

./tc sample024.mpl

ERROR(3): ')' is not found in output_statement
Check grammar of 3 line in sample024.mpl

./tc sample24.mpl
Name Type Def. Ref.
./tc sample25.mpl
Name Type Def. Ref.
ch char 4 7, 8,
./tc sample025.mpl

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

./tc sample025t.mpl

ERROR(8): Keyword 'then' is not found in condition_statement
Check grammar of 8 line in sample025t.mpl

./tc sample25t.mpl
Name Type Def. Ref.
ch char 3 5, 8,
int integer 3 5, 8,
boolx integer 3 5, 8,
booly boolean 3 5, 9,
./tc sample26.mpl
Name Type Def. Ref.
n 3 8, 12,
i 4 8, 11,
sum integer 5 8, 12,
./tc sample026.mpl
Name Type Def. Ref.
n 2 4, 11,
i 2 5, 9,
sum integer 2 6, 11,
./tc sample026t.mpl

ERROR(9): Keyword 'do' is not found in iteration_statement
Check grammar of 9 line in sample026t.mpl

```

```
./tc sample27.mpl
Name Type Def. Ref.
i 2 4, 15,
j 2 6, 13,
k integer 2 8, 11,
./tc sample28p.mpl
Name Type Def. Ref.
./tc sample028p.mpl

ERROR(4): Procedure name is not found in call_statement
Check grammar of 4 line in sample028p.mpl

./tc sample028q.mpl

ERROR(4): ')' is not found in call_statement
Check grammar of 4 line in sample028q.mpl

./tc sample028r.mpl

ERROR(4): ')' is not found in call_statement
Check grammar of 4 line in sample028r.mpl

./tc sample29p.mpl
Name Type Def. Ref.
unused1 integer 9
UnusedArrayForTestinteger 10
n integer 11
gc:n integer 11 23, 24,
lc:n integer 11 24,
a:n integer 14 16, 23,
b:n integer 14 17, 21,
r:n integer 14 19, 21,
b integer 26
b integer 30
gc:b integer 30 33, 45,
lc:b integer 31 36, 45,
aa:b integer 31 35, 45,
bb:b integer 31 35, 45,
b integer 40
unusedchar char 48
a2 integer 49
gc:a2 integer 50 55, 57,
x2 integer 60 111, 134,
y1:x2 integer 60 65, 90,
y2:x2 integer 60 63, 90,
lc:x2 integer 61 63, 67,
y11:x2 integer 61 65, 84,
x2 integer 71
x2 integer 75
gc:x2 integer 76 78, 83,
y22:x2 integer 76 80, 85,
x2 integer 88
unusedarray integer 92
b integer 93
b integer 99
x1 integer 106 111, 134,
y1 integer 106 115, 120,
y2 integer 106 115, 120,
com char 107 115, 121,
endflag boolean 108 112, 133,
./tc sample029p.mpl

ERROR(5): Variable name is not found in variable
Check grammar of 5 line in sample029p.mpl

./tc sample029q.mpl

ERROR(6): ')' is not found in input_statement
Check grammar of 6 line in sample029q.mpl

./tc sample029r.mpl
File sample029r.mpl can not open.
./tc sample029s.mpl

ERROR(6): Number is not found in output_format
Check grammar of 6 line in sample029s.mpl

./tc sample029t.mpl

ERROR(7): Keyword 'end' is not found in compound_statement
Check grammar of 7 line in sample029t.mpl

```

```
test1.mpl

ERROR(1): Keyword 'program' is not found
Check grammar of 1 line in test1.mpl

test2.mpl

ERROR(1): Program name is not found
Check grammar of 1 line in test2.mpl

test3.mpl

ERROR(1): ';' is not found in parse_program
Check grammar of 1 line in test3.mpl

test4.mpl

ERROR(1): Variable name is not found in variable_names1
Check grammar of 1 line in test4.mpl

test5.mpl

ERROR(1): Variable name is not found in variable_names1
Check grammar of 1 line in test5.mpl

test6.mpl

ERROR(1): ':' is not found in variable_declaration1
Check grammar of 1 line in test6.mpl

test7.mpl

ERROR(1): Type is not found in type
Check grammar of 1 line in test7.mpl

test8.mpl

ERROR(1): ';' is not found in variable_declaration
Check grammar of 1 line in test8.mpl

test9.mpl

ERROR(1): ':' is not found in variable_declaration2
Check grammar of 1 line in test9.mpl

test10.mpl

ERROR(1): Type is not found in type
Check grammar of 1 line in test10.mpl

test11.mpl

ERROR(1): ';' is not found in variable_declaration
Check grammar of 1 line in test11.mpl

test12.mpl

ERROR(2): Keyword 'begin' is not found
Check grammar of 2 line in test12.mpl

test13.mpl

ERROR(2): Keyword 'end' is not found in compound_statement
Check grammar of 2 line in test13.mpl

test14.mpl

ERROR(2): '.' is not found at the end of program
Check grammar of 2 line in test14.mpl

test15.mpl
Name Type Def. Ref.
a integer 1
test16.mpl

ERROR(1): Variable name is not found in variable_names2
Check grammar of 1 line in test16.mpl

tb1.mpl

ERROR(1): Procedure_name is not found in subprogram_declaration
Check grammar of 1 line in tb1.mpl

tb2.mpl

```

```
ERROR(1): ';' is not found in subprogram_declaration
Check grammar of 1 line in tb2.mpl

tb3.mpl

ERROR(1): Variable name is not found in variable_names1
Check grammar of 1 line in tb3.mpl

tb4.mpl

ERROR(1): ':' is not found in formal_parameters
Check grammar of 1 line in tb4.mpl

tb5.mpl

ERROR(1): Type is not found in type
Check grammar of 1 line in tb5.mpl

tb6.mpl
comand.sh: line 105: 29288 Segmentation fault (core dumped) ./tc tb6.mpl
tb7.mpl
comand.sh: line 107: 29289 Segmentation fault (core dumped) ./tc tb7.mpl
tb8.mpl
comand.sh: line 109: 29290 Segmentation fault (core dumped) ./tc tb8.mpl
tb9.mpl
comand.sh: line 111: 29291 Segmentation fault (core dumped) ./tc tb9.mpl
tb10.mpl
comand.sh: line 113: 29292 Segmentation fault (core dumped) ./tc tb10.mpl
tb11.mpl
comand.sh: line 115: 29293 Segmentation fault (core dumped) ./tc tb11.mpl
tb12.mpl
comand.sh: line 117: 29294 Segmentation fault (core dumped) ./tc tb12.mpl
tb13.mpl
comand.sh: line 119: 29295 Segmentation fault (core dumped) ./tc tb13.mpl
tb14.mpl
comand.sh: line 121: 29296 Segmentation fault (core dumped) ./tc tb14.mpl
tb15.mpl
comand.sh: line 123: 29297 Segmentation fault (core dumped) ./tc tb15.mpl
tb16.mpl
comand.sh: line 125: 29298 Segmentation fault (core dumped) ./tc tb16.mpl
tb17.mpl
comand.sh: line 127: 29299 Segmentation fault (core dumped) ./tc tb17.mpl
tb18.mpl

ERROR(1): ';' is not found in subprogram_declaration
Check grammar of 1 line in tb18.mpl

tb19.mpl
comand.sh: line 131: 29301 Segmentation fault (core dumped) ./tc tb19.mpl
tb20.mpl

ERROR(2): ')' is not found in factor
Check grammar of 2 line in tb20.mpl

tb21.mpl

ERROR(2): ')' is not found in factor
Check grammar of 2 line in tb21.mpl

tb22.mpl

ERROR(2): '(' is not found in factor
Check grammar of 2 line in tb22.mpl

tb23.mpl

ERROR(2): Factor is not found in factor
Check grammar of 2 line in tb23.mpl

tt1.mpl

ERROR(1): '[' is not found in array_type
Check grammar of 1 line in tt1.mpl

tt2.mpl

ERROR(1): Number is not found in array_type
Check grammar of 1 line in tt2.mpl

tt3.mpl

ERROR(1): ']' is not found in array_type
Check grammar of 1 line in tt3.mpl
```

```
tt4.mpl
ERROR(1): Keyword 'of' is not found in array_type
Check grammar of 1 line in tt4.mpl
tt5.mpl
ERROR(1): Standard type is not found in standard_type
Check grammar of 1 line in tt5.mpl
tt6.mpl
ERROR(1): ')' is not found in formal_parameters
Check grammar of 1 line in tt6.mpl
tt7.mpl
ERROR(1): ';' is not found in subprogram_declaration
Check grammar of 1 line in tt7.mpl
ta1.mpl
ERROR(2): ':=' is not found in return_statement
Check grammar of 2 line in ta1.mpl
ta2.mpl
ERROR(2): Factor is not found in factor
Check grammar of 2 line in ta2.mpl
ta3.mpl
ERROR(2): Factor is not found in factor
Check grammar of 2 line in ta3.mpl
ta4.mpl
ERROR(2): Factor is not found in factor
Check grammar of 2 line in ta4.mpl
ta5.mpl
ERROR(2): ']' is not found in variable
Check grammar of 2 line in ta5.mpl
ta6.mpl
ERROR(2): ':=' is not found in return_statement
Check grammar of 2 line in ta6.mpl
ta7.mpl
ERROR(2): Factor is not found in factor
Check grammar of 2 line in ta7.mpl
ta8.mpl
ERROR(2): Keyword 'end' is not found in compound_statement
Check grammar of 2 line in ta8.mpl
ta9.mpl
ERROR(2): Keyword 'end' is not found in compound_statement
Check grammar of 2 line in ta9.mpl
ta10.mpl
ERROR(2): Keyword 'end' is not found in compound_statement
Check grammar of 2 line in ta10.mpl
ta11.mpl
ERROR(2): Keyword 'end' is not found in compound_statement
Check grammar of 2 line in ta11.mpl
ta12.mpl
ERROR(2): Keyword 'end' is not found in compound_statement
Check grammar of 2 line in ta12.mpl
tm1.mpl
Name Type Def. Ref.
a integer 3
tm2.mpl
```

```
411
412 ERROR(2): string is too long
413 Check grammar of 2 line in am2.mpl
414
415 tm3.mpl
416 Name Type Def. Ref.
417 b integer 3 4,
418 a integer 3
419 tm4.mpl
420
421 ERROR(4): number is too long
422 Check grammar of 4 line in tm4.mpl
423
424 sample31p.mpl
425 Name Type Def. Ref.
426 a integer 2 3, 22,
427 b char 8 20, 21,
428 a boolean 10
429 q:a integer 11 14, 15,
430 c integer 18
431 sample032p.mpl
432 Name Type Def. Ref.
433 sample33p.mpl
434 Name Type Def. Ref.
435 a 2 3, 25,
436 x integer 2 5, 9,
437 b char 11 23, 24,
438 a boolean 13
439 q:a integer 14 17, 18,
440 c integer 21
441 sample34.mpl
442 Name Type Def. Ref.
443 num integer 2 6, 18,
444 ch char 2 6, 17,
445 sample35.mpl
446 Name Type Def. Ref.
447 i integer 2 4, 14,
448 b boolean 2
449 c char 2
```

4.3　テストデータの十分性

コード 7.1 を実行したときの結果 (コード 7.3) みたとき、

- Lines executed:86.25% of 160
- Branches executed:95.74% of 94
- Taken at least once:82.98% of 94

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

c0 カバレッジと s0 カバレッジの網羅率は 86% 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 95.74% であったが、全条件分岐の網羅率は 82.98% となった。

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
12/01	12/01	1	(a)	スケジュールを立てる
12/01	12/01	0.5	(b-1)	配布された資料を読み直す
12/01	12/01	0.5	(b-2)	配布されたプログラムを読む
12/01	12/01	1	(b-3)	コンパイラのテキスト (プログラム) を読む
12/4	12/4	5	(c)	字句解析系の概略設計
12/4	12/4	2	(e-1-1)	ブラックボックステスト用プログラムの作成
12/4	12/4	5	(d-4)	解析器の作成
12/4	12/4	1	(e-1-2)	バグがない場合の想定テスト結果の準備
12/11	12/11	1	(d-3)	カウントした結果の出力部分の作成
12/11	12/11	0.5	(e-2-1)	カバレッジレベルの決定
12/11	12/11	2	(e-2-2)	ホワイトボックステスト用プログラムの作成
12/11	12/11	1	(e-2-3)	バグがない場合の想定テスト結果の準備
12/15	12/15	8	(f)	テストとデバッグを行う
12/15	12/15	1	(g-1)	作成したプログラムの設計情報を書く
12/15	12/15	1	(g-2)	テスト情報を書く
12/17	12/17	1	(g-3)	事前計画と実際の進捗状況を書く
12/17	12/17	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

前回は理想像に基づいた計画をたて、序盤で計画通り進まなくなった。今回は、自分の計画も考慮しつつ計画を立てた。

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
12/01	12/01	1	(a)	12/01	0.5
12/07	12/01	0.5	(b-1)	12/1	1
12/07	12/01	0.5	(b-2)	12/1	1
12/07	12/01	1	(b-3)	12/1	1
12/14	12/4	5	(c)	12/4	3
12/14	12/4	2	(e-1-1)	12/4	1
12/14	12/4	5	(d-4)	12/4	1
12/14	12/4	1	(e-1-2)	12/6	5
12/11	12/11	1	(d-3)	12/16	1
12/11	12/11	0.5	(e-2-1)	12/16	1
12/11	12/11	2	(e-2-2)	12/16	1
12/11	12/11	1	(e-2-3)	12/16	1
12/15	12/15	8	(f)	12/17	5
12/15	12/15	1	(g-1)	12/18	1
12/15	12/15	1	(g-2)	12/18	1
12/19	12/19	1	(g-3)	12/19	1
12/19	12/19	5	(h)	12/29	10

5.4　当初の事前計画と実際の進捗との差の原因

表 2 より進行に差があった。バージョン管理のため git を導入したが、上手く保存ができておらず、消えてしまったデータを再度つくる手間がかかった。

6　ソースコード

ソースコード 74　pprinter-list.c

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

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

ソースコード 75　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 /* token-list.c */
106 extern void init_idtab();
107 extern void id_countup(char *np);
108 extern void print_idtab();
109 extern void release_idtab();
110
111 /* pprinter-list.c */
112 extern void error(char *mes);
113
114 /* ebnf-list.c */
115 extern void set_token(int token);
116 extern int parse_program();
117 extern int parse_block();
118 extern int parse_variable_declaration();
119 extern int parse_variable_names();
120 extern int parse_variable_name();
121 extern int parse_type();
122 extern int parse_standard_type();
123 extern int parse_array_type();
124 extern int parse_subprogram_declaration();
125 extern int parse_procedure_name();
126 extern int parse_formal_parameters();
127 extern int parse_compound_statement();
128 extern int parse_statement();
129 extern int parse_condition_statement();
130 extern int parse_iteration_statement();
131 extern int parse_exit_statement();
132 extern int parse_call_statement();
133 extern int parse_expressions();
134 extern int parse_return_statement();
135 extern int parse_assignment_statement();
136 extern int parse_left_part();
137 extern int parse_variable();
138 extern int parse_expression();
139 extern int parse_simple_expression();
140 extern int parse_term();
141 extern int parse_factor();
142 extern int parse_constant();
143 extern int parse_multiplicative_operator();
144 extern int parse_additive_operator();
145 extern int parse_relational_operator();
```

```
146 extern int parse_input_statement();
147 extern int parse_output_statement();
148 extern int parse_output_format();
149 extern int parse_empty_statement();
```

ソースコード 76 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
};
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
                        cbuf = fgetc(fp);
```

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

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

```
146                     num_attr = num_attr * 10 + (cbuf - 48);
147                     cbuf = fgetc(fp);
148                     if ((num_attr > MAXINTSIZE) || (num_attr < 0))
149                     {
150                             error("number is too long\n");
151                             return ERROR;
152                     }
153             }
154             // cbuf = fgetc(fp);
155             printf(" %d", num_attr);
156             return TNUMBER;
157     }
158     else
159     {
160             int i = TPLUS;
161             for (i = TPLUS; i <= TSEMI; i++)
162             {
163                     if (cbuf == tokenstr[i][0])
164                     {
165                             if (i == TNOTEQ)
166                             { // if cbuf is '<' check whether '<>' , '<=' or '<'
167                                     return noteqPrint();
168                             }
169                             else if (i == TGR)
170                             { // if cbuf is '>' check whether '>=' or '>'
171                                     return grPrint();
172                             }
173                             else if (i == TASSIGN)
174                             { // if cbuf is ':' check whether ':=' or ':'
175                                     return assignPrint();
176                             }
177                             else if (i == TSEMI)
178                             {
179                                     return semiPrint();
180                             }
181                             else
182                             {
183                                     cbuf = fgetc(fp);
184                                     printf(" %s", tokenstr[i]);
185                                     return i;
186                             }
187                     }
188             }
189             if (cbuf == EOF)
190             {
191                     return EOFCODE;
192             }
193
194             char dst[100];
```

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

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

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

```
342 { // print process when token is program
343         print_indent();
344         printf(" %s", string_attr);
345         init_string_atr(count);
346         num_indent += 1;
347         return TPROGRAM;
348 }
349
350 int beginPrint(int count)
351 { // print process when token is begin
352         if (prestate == THEN)
353         {
354                 num_then--;
355                 num_indent--;
356                 printf("\r");
357                 print_tab();
358         }
359         else if (prestate != SEMI)
360         {
361                 print_indent();
362         }
363         init_string_atr(count);
364         printf(" %s", string_attr);
365         num_indent += 1;
366         print_indent();
367         return TBEGIN;
368 }
369
370 int endPrint(int count)
371 { // print process when token is end
372         if (prestate == SEMI)
373         {
374                 if (num_then > 0)
375                 {
376                         num_then -= 1;
377                         printf("\r");
378                         num_indent -= 2;
379                         print_tab();
380                         printf(" %s", string_attr);
381                         init_string_atr(count);
382                         return TEND;
383                 }
384                 printf("\r");
385                 num_indent -= 1;
386                 print_tab();
387                 printf(" %s", string_attr);
388                 init_string_atr(count);
389                 return TEND;
390         }
```

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

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

```
489         {
490                 printf("%s", tokenstr[TGR]);
491                 return TGR;
492         }
493 }
494
495 int assignPrint(void)
496 {
497         cbuf = fgetc(fp);
498         if (cbuf == '=')
499         {
500                 cbuf = fgetc(fp);
501                 printf(" %s", tokenstr[TASSIGN]);
502                 return TASSIGN;
503         }
504         else
505         {
506                 printf("%s", tokenstr[TCOLON]);
507                 return TCOLON;
508         }
509 }
510
511 int semiPrint(void)
512 {
513         prestate = SEMI;
514         cbuf = fgetc(fp);
515         printf("%s", tokenstr[TSEMI]);
516         print_indent();
517         return TSEMI;
518 }
```

ソースコード 77 ebnf-list.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 78 test1.mpl

```
1 aaa
```

ソースコード 79 test2.mpl

```
1 program
```

ソースコード 80 test3.mpl

```
1 program aa
```

ソースコード 81 test4.mpl

```
1 program aa; var
```

ソースコード 82 test05.mpl

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

ソースコード 83 test6.mpl

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

ソースコード 84 test7.mpl

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

ソースコード 85 test8.mpl

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

ソースコード 86 test9.mpl

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

ソースコード 87 test10.mpl

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

ソースコード 88 test11.mpl

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

ソースコード 89 test12.mpl

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

ソースコード 90 test13.mpl

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

ソースコード 91 test14.mpl

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

ソースコード 92 test15.mpl

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

ソースコード 93 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 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
16 ;var a: integer; a: integer;
17 begin end.
```

ソースコード 94 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 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
16 ;var a: integer; a: integer;
17 begin end.
```

ソースコード 95　tm3.mpl

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

ソースコード 96　tm4.mpl

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

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