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

作成するプログラム名を mpplc, MPPL で書かれたプログラムのファイル名を foo.mpl とするとき foo.mpl の内容のプログラムをコンパイルした結果（CASL II のプログラム）を持つファイル foo.csl を生成するプログラムを作成する

3　プログラムの設計情報

ここにはプログラムの設計情報を記述する

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

■Compiler4-submit.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-list2.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() を呼び出し、token を解析する。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() 関数は呼び出され実行される。

■id-list2.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-list2.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　char *GetTokenString() 関数

■機能　scan 関数で収得した変数名を返す。

■引数　なし.

■戻り値　scan 関数で収得した変数名を返す。

ソースコード 5　search_idtab(char *np)

```
1  char *GetTokenString()
2  {
3    return token_string;
4  }
```

3.4.2　int GetTokenNum() 関数

■機能　scan 関数の数値を返す。

■引数　なし.

■戻り値　scan 関数の数値を返す。

ソースコード 6　GetTokenNum

```
1  int GetTokenNum()
2  {
3    return num_attr;
4  }
```

3.4.3　char *getTokenProcname(char *np) 関数

■機能　scan 関数で習得した関数名を返す。

■引数　変数名.

■戻り値　変数名をしらべ、関数ないにある変数ならば、関数名を返す。なければヌル文字を返す。

ソースコード 7　*getTokenProcname(char *np)

```
1  char *getTokenProcname(char *np)
2  {
3    struct ID *p;
4    p = search_idtab(np);
5    if (p == NULL)
6    {
7      struct ID *q;
8      for (q = head; q != NULL; q = q->nextp)
9      {
10       printf("name:%s, procname:%s, count:%d\n", q->name, q->procname, q->count);
```

```
11        }
12        char buf[MAXSTRSIZE];
13        sprintf(buf, "can not find the name :%s(%s)\n", np, subprogramname);
14        error(buf);
15        return NULL;
16      }
17      else
18      {
19        if (strcmp(p->procname, ""))
20        {
21          return "";
22        }
23        else
24        {
25          return p->procname;
26        }
27      }
28    }
```

3.4.4　void initfile(char *filename) 関数

■機能　出力ファイルの初期化を行う。

■引数　出力ファイル名.

■戻り値　なし.

ソースコード 8　sinitfile(char *filename)

```
1  void initfile(char *filename)
2  {
3      fp2 = fopen(filename, "w");
4  }
```

3.4.5　void closefile() 関数

■機能　出力ファイルを閉じる。

■引数　なし.

■戻り値　なし.

ソースコード 9　closefile()

```
1  void closefile()
2  {
3      fclose(fp2);
4  }
```

3.4.6　int get_new_label_num(void) 関数

■機能　あたらしいラベル番号を返す。

■引数　なし.

■戻り値　ラベル番号の数値.

ソースコード 10　get_new_label_num)

```
1  int get_new_label_num(void)
2  {
3      static int labelcounter = 1;
4      return labelcounter++;
5  }
```

3.4.7　char *get_label() 関数

■機能　今あるラベル番号を返す。

■引数　なし.

■戻り値　L からはじまるラベル番号の文字列.

ソースコード 11　*get_label()

```
1  char *get_label(int label)
2  {
3      char buf[MAXSTRSIZE];
4      char *cslfp = buf;
5      sprintf(cslfp, "L%04d", label);
6      return cslfp;
7  }
```

3.4.8　void gen_code(char *code) 関数

■機能　インデントつきの出力ファイルに code を出力する。

■引数　出力する文字列.

■戻り値　なし.

,label=code:gen_code1(char *code)] void gen_code(char *code) char cslfp[100]; sprintf(cslfp, ”printf(”fprintf(fp2, ”

3.4.9　void gen_codeNoindent(char *code) 関数

■機能　インデントなしの出力ファイルに code を出力する。

■引数　出力する文字列.

■戻り値　なし.

ソースコード 12　gen_codeNoindent(char *code)

```
1  void gen_codeNoindent(char *code)
2  {
3      char cslfp[100];
4      sprintf(cslfp, "%s\n", code);
5      printf("%s", cslfp);
6      fprintf(fp2, "%s\n", code);
7  }
```

3.4.10　void gen_code_label(char *code, int label) 関数

■機能　ラベル文字つきの出力ファイルに code を出力する。

■引数　出力する文字列. ラベル番号の数値

■戻り値　なし.

ソースコード 13　gencodelabel()

```
1  void gen_code_label(char *code, int label)
2  {
3      char cslfp[100];
4      sprintf(cslfp, "\t%s\tL%04d\n", code, label);
5      printf("%s", cslfp);
6      fprintf(fp2, "\t%s\tL%04d\n", code, label);
7  }
```

3.4.11　void gen_label(int label) 関数

■機能　ラベル文字つきの出力ファイルに code を出力する。

■引数　ラベル番号の数値

■戻り値　なし.

ソースコード 14　gen_label1(int label)

```
1  void gen_label(int label)
2  {
3      char cslfp[100];
4      sprintf(cslfp, "L%04d\tDS\t0\n", label);
5      printf("%s", cslfp);
6      fprintf(fp2, "L%04d\tDS\t0\n", label);
7  }
```

3.4.12　char *GetIDName() 関数

■機能　一番最後に登録された ID の名前を返す。

■引数　なし

■戻り値　一番最後に登録された ID の名前を返す. なければヌル文字を返す。

ソースコード 15　*GetIDName()

```
1  char *GetIDName()
2  {
3    return tail->name;
4  }
```

3.4.13　char *GetParameterName() 関数

■機能　一番最後に登録された ID の関数名を返す。

■引数　なし

■戻り値　一番最後に登録された ID の関数名を返す. なければヌル文字を返す。

ソースコード 16　*GetParameterName()

```
1  char *GetParameterName()
2  {
3    return subprogramname;
4  }
```

3.4.14　int GetIDType() 関数

■機能　一番最後に登録された ID の型を返す。

■引数　なし

■戻り値　一番最後に登録された ID の型を返す. なければヌル文字を返す。

ソースコード 17　GetIDType()

```
1  int GetIDType()
2  {
3    return tail->ttype;
4  }
```

3.4.15　void Showparse_program(int label1) 関数

■機能　プログラムの文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.16　void Showparse_program2(int label1) 関数

■機能　プログラムの文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.17　void Showparse_variable_declaration() **関数**

■機能　変数宣言の文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.18　void Showparse_subprogram_declaration(bool isParameter) **関数**

■機能　サブプログラム宣言の文法解析中の出力を行う。

■引数　パラメータの宣言かどうかの真偽値

■戻り値　なし

3.4.19　void Showparse_formal_parameters() **関数**

■機能　仮引数の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.20　void Showparse_compound_statement() **関数**

■機能　複合文の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.21　void Showparse_condition_statement(int label1) **関数**

■機能　条件文の文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.22　void Showparse_condition_statement2(int label1, int label2) **関数**

■機能　条件文の文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.23　void Showparse_iteration_statement(int label1) **関数**

■機能　繰り返し文の文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.24　void Showparse_iteration_statement2(int label2) **関数**

■機能　繰り返し文の文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.25　void Showparse_iteration_statement3(int label1, int label2) **関数**

■機能　繰り返し文の文法解析中の出力を行う。

■引数　ラベル番号の数値

■戻り値　なし

3.4.26　void Showparse_assignment_statement() **関数**

■機能　代入文の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.27　void Showparse_expression() **関数**

■機能　式の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.28　void Showparse_simple_expression(int opr) **関数**

■機能　単純式の文法解析中の出力を行う。

■引数　scan 関数で収得した token

■戻り値　なし

3.4.29　void Showparse_term(int opr) **関数**

■機能　項の文法解析中の出力を行う。

■引数　scan 関数で収得した token

■戻り値　なし

3.4.30　void Showparse_factor() 関数

■機能　因子の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.31　void Showparse_factor2() 関数

■機能　因子の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.32　void Showparse_input_statement() 関数

■機能　入力文の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.33　void Showparse_input_statement2() 関数

■機能　入力文の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.34　void Showparse_output_statement() 関数

■機能　出力文の文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.35　void Showparse_output_format() 関数

■機能　出力フォーマットの文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.4.36　void Showparse_output_format2() 関数

■機能　出力フォーマットの文法解析中の出力を行う。

■引数　なし

■戻り値　なし

3.5　各関数の外部仕様 (第三回作成したもの)

3.5.1　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 18　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 }
```

3.5.2　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 19　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  }
```

3.5.3　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 20　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  }
```

3.5.4　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 21　setDefLine()

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

3.5.5　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 22　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 }
```

3.5.6　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 23　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 }
```

3.6　各関数の外部仕様 (課題 2)

3.6.1　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 24　init_string_atr()

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

3.6.2　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 25　set_token()

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

3.6.3　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 26　parse_program()

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

3.6.4　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 27　parse_block()

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

3.6.5　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 28　parse_variable_declaration()

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

3.6.6　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 29　parse_variable_names()

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

3.6.7　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 30　parse_type()

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

3.6.8　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 31　parse_standard_type()

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

3.6.9　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 32　parse_array_type()

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

3.6.10　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 33　parse_subprogram_declaration()

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

3.6.11　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 34　parse_formal_parameters()

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

3.6.12　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 35　parse_compound_statement()

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

3.6.13　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 36　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     */
```

3.6.14　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 37　parse_condition_statement()

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

3.6.15　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 38　parse_iteration_statement()

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

3.6.16　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 39　parse_exit_statement()

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

3.6.17　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 40　parse_call_statement()

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

3.6.18　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 41　parse_expressions()

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

3.6.19　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 42　parse_return_statement()

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

3.6.20　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 43　parse_assignment_statement()

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

3.6.21　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 44　parse_variable()

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

3.6.22　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 45　parse_expression()

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

3.6.23　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 46　parse_simple_expression()

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

3.6.24　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 47　parse_term()

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

3.6.25　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 48　parse_factor()

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

3.6.26　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 49　parse_constant()

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

3.6.27　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 50　parse_multiplicative_operator()

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

3.6.28　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 51　parse_input_statement()

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

3.6.29　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 52　parse_output_statement()

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

3.6.30　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 53　parse_output_format()

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

3.6.31　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 54　parse_empty_statement()

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

3.6.32　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 55　programPrint

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

3.6.33　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 56　beginPrint

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

3.6.34　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 57　ifPrint

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

3.6.35　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 58　elsePrint

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

3.6.36　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 59　thenPrint

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

3.6.37　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 60　noteqPrint

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

3.6.38　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 61　grPrint

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

3.6.39　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 62　assignPrint

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

3.6.40　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 63　semiPrint

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

3.6.41　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 64　endPrint

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

3.7　各関数の外部仕様 (課題 1)

3.7.1　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 65　init_scan()

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

3.7.2　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 66　scan()

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

3.7.3　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 67　get_linenum()

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

3.7.4　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 68　end_scan()

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

3.7.5　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 69　isChar()

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

3.7.6　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 70　UntilFun()

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

3.7.7　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 71　UntilComment()

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

3.7.8　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 72　UntilString()

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

3.7.9　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 73　init_idtab()

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

3.7.10　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 74　id_countup()

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

3.7.11　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 75　print_idtab()

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

3.7.12　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 76　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

- sample01.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 77　演 sampl31p.mpl の実行例

```
$ gcc −o tc Compiler4_submit.c scan−list.c id−list2.c cross−main.h ebnf−list2.c
$ ./tc sample01.mpl
```

sample01.csl というファイルができ、結果は以下のようになった。

ソースコード 78　result

```
1   $$sample11pp START
2           LAD gr0,0
3           CALL L0001
4           CALL FLUSH
5           SVC 0
6   $n%kazuyomikomi DC 0
7   $kazuyomikomi DS 0
8           POP gr2
9           POP gr1
10          ST gr1, $n%kazuyomikomi
11          PUSH 0,gr2
12          LAD gr1, L0002
13          LD gr2,gr0
14          CALL WRITESTR
15          CALL WRITELINE
16          LD gr1, $n%kazuyomikomi
17          CALL READINT
18          CALL READLINE
19          RET
20  $sum DC 0
21  $wakakidasi DS 0
22          LAD gr1, L0003
23          LD gr2,gr0
24          CALL WRITESTR
25          LD gr1, $sum
26          LD gr2,gr0
27          CALL WRITEINT
28          CALL WRITELINE
29          RET
30  $data DC 0
31  $s%goukei DC 0
32  $n%goukei DC 0
33  $data%goukei DC 0
34  $goukei DS 0
35          POP gr2
```

```
36              POP gr1
37              ST gr1, $s%goukei
38              POP gr1
39              ST gr1, $n%goukei
40              PUSH 0,gr2
41              LD gr1, $s%goukei
42              PUSH 0,gr1
43              LAD gr1, 0
44              POP gr2
45              ST gr1,0,gr2
46      L0004 DS 0
47              LD gr1, $n
48              LD gr1,0,gr1
49              PUSH 0, gr1
50              LAD gr1, 0
51              POP gr2
52              CPA gr2,gr1
53              JPL L0006
54              LD gr1,gr0
55              JUMP L0007
56      L0006 DS 0
57              LAD gr1, 1
58      L0007 DS 0
59              CPA gr1,gr0
60              JZE L0005
61              LD gr1, $data%goukei
62              CALL READINT
63              CALL READLINE
64              LD gr1, $s%goukei
65              PUSH 0,gr1
66              LD gr1, $s
67              LD gr1,0,gr1
68              PUSH 0, gr1
69              LD gr1, $data
70              POP gr2
71              ADDA gr1, gr2
72              JOV EOVF
73              POP gr2
74              ST gr1, 0, gr2
75              LD gr1, $n%goukei
76              PUSH 0,gr1
77              LD gr1, $n
78              LD gr1,0,gr1
79              PUSH 0, gr1
80              LAD gr1, 1
81              POP gr2
82              SUBA gr1, gr2
83              JOV EOVF
84              LD gr1, gr2
```

```
85              POP gr2
86              ST gr1, 0, gr2
87              JUMP L0004
88      L0005 DS 0
89              RET
90      $n DC 0
91              LD gr1, $n
92              CALL $kazuyomikomi
93              LD gr1, $n
94              LAD gr1, 2
95              POP gr2
96              POP gr1
97              MULA gr1,gr2
98              PUSH 0,gr1
99              LD gr1, $sum
100             CALL $goukei
101             CALL $wakakidasi
102             RET
```

sample01.mpl は 79 である。

ソースコード 79　sample1p.mpl

```
1   program sample11pp;
2   procedure kazuyomikomi(n : integer);
3   begin
4       writeln('input the number of data');
5       readln(n)
6   end;
7   var sum : integer;
8   procedure wakakidasi;
9   begin
10      writeln('Sum of data = ', sum)
11  end;
12  var data : integer;
13  procedure goukei(n, s : integer);
14      var data : integer;
15  begin
16      s := 0;
17      while n > 0 do begin
18              readln(data);
19              s := s + data;
20              n := n − 1
21      end
22  end;
23  var n : integer;
24  begin
25      call kazuyomikomi(n);
26      call goukei(n ∗ 2, sum);
27      call wakakidasi
```

```
28    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 80　a を 1024 個書いたときの結果

```
1    program aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
2          ......
3        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaatm1.mpl;
4         var a: integer;
5         a: integer;
6         begin
7
8         end .
```

ソースコード 81　a を 1025 個書いたときの結果

```
1    ERROR(2): string is too long
2    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 82　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 83　2,147,483,648 書いたときの結果

```
1    ERROR(4): number is too long
2    Check grammar of 4 line in tm4.mpl
```

4.3　テストデータの十分性

テストを実行したとき、分岐率は以下のようになった。

- executed:87.93% of 116
- Branches executed:96.67% of 60
- Taken at least once:83.33% of 60

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 87% 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 96% であったが、全条件分岐の網羅率は 82% となった。動的に領域を確保するのに失敗したとき、エラーを返す分岐処理が通ることができず、分岐率は 100% にできなかった。

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

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
1/01	1/01	1	(a)	1/01	0.5
1/07	1/01	0.5	(b-1)	1/1	1
1/07	1/01	0.5	(b-2)	1/1	1
1/07	1/01	1	(b-3)	1/1	1
1/14	1/4	5	(c)	1/4	3
1/14	1/4	2	(e-1-1)	1/4	1
1/14	1/4	5	(d-4)	1/4	1
1/14	1/4	1	(e-1-2)	1/6	5
1/11	1/11	1	(d-3)	1/16	1
1/11	1/11	0.5	(e-2-1)	1/16	1
1/11	1/11	2	(e-2-2)	1/16	1
1/11	1/11	1	(e-2-3)	1/16	1
1/15	1/15	8	(f)	1/17	5
1/15	1/15	1	(g-1)	1/18	1
1/15	1/15	1	(g-2)	1/18	1
1/19	1/19	1	(g-3)	1/19	1
1/19	1/19	5	(h)	1/30	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。アセンブリ言語の読み方や使い方等覚えるのに時間がかかった。

6 ソースコード

ソースコード 84 test1.mpl

```
1 aaa
```

ソースコード 85 test2.mpl

```
1 program
```

ソースコード 86 test3.mpl

```
1 program aa
```

ソースコード 87 test4.mpl

```
1 program aa; var
```

ソースコード 88 test05.mpl

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

ソースコード 89 test6.mpl

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

ソースコード 90 test7.mpl

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

ソースコード 91 test8.mpl

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

ソースコード 92 test9.mpl

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

ソースコード 93 test10.mpl

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

ソースコード 94 test11.mpl

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

ソースコード 95 test12.mpl

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

ソースコード 96 test13.mpl

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

ソースコード 97 test14.mpl

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

ソースコード 98 test15.mpl

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

ソースコード 99 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 100 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  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
16  ;var a: integer; a: integer;
17  begin end.
```

ソースコード 101　tm3.mpl

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

ソースコード 102　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