目次

1 演習の目的 3

2 演習内容 3

3 プログラムの設計情報 3

3.1 全体構成 . . . 3

3.2 各モジュールごとの構成 . . . 3

3.3 各関数の外部仕様 . . . 5

3.3.1 init_scan 関数 . . . 5

3.3.2 scan 関数 . . . 5

3.3.3 get_linenum 関数 . . . 5

3.3.4 end_scan 関数 . . . 5

3.3.5 isNumber 関数 . . . 6

3.3.6 isChar 関数 . . . 6

3.3.7 UntilFun 関数 . . . 6

3.3.8 UntilComment 関数 . . . 6

3.3.9 UntilString 関数 . . . 7

3.3.10 init_idtab 関数 . . . 7

3.3.11 id_countup 関数 . . . 7

3.3.12 print_idtab 関数 . . . 7

3.3.13 release_idtab 関数 . . . 7

4 テスト情報 8

4.1 テストデータ . . . 8

4.2 テスト結果 . . . 10

4.3 テストデータの十分性 . . . 14

5 事前計画と実際の進捗状況 14

5.1 事前計画 . . . 14

5.2 事前計画の立て方についての前課題からの改善点 . . . 15

5.3 実際の進捗状況 . . . 15

5.4 当初の事前計画と実際の進捗との差の原因 . . . 16

6 ソースコード 17

7 参考文献 29

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 を読み込み，字句 (トークン) がそれぞれ何個出現したかを数え，出力するプログラムを作成した.

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

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

■id-list.c
名前を実体ごとに出現個数を数えあげるモジュール。id_countup() 関数は scan-list.c からトークン NAME を拾得したとき呼び出される。print_idtab() は token-list.c の画面に出力時に呼び出される。release_idtab は token-list.c の終了時に呼び出される。

■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 token-list.c から呼び出され実行される。

■token-list.h
定数トークンや関数の宣言を行うモジュール。定数トークンは scan-list.c や token-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。

■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 を持つ。そして nextp が次の格納場所 (アドレス) を保持する。

ソースコード 1　構造体 ID in id-list.c

```
1  struct ID {
2      char *name;
3      int count;
4      struct ID *nextp;
5  } *idroot;
```

■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 2　構造体 KEY in token-list.h

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

■変数

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

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

int token　scan 関数で収得した TOKEN を保持する変数

ソースコード 3　各変数 in token-list.h

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

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.h

```
1  int cbuf, num_line = 0;
```

```
2     int num_attr;
3     char string_attr[MAXSTRSIZE];
4     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。

3.3.1　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 5　init_scan()

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

3.3.2　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 6　scan()

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

3.3.3　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 7　get_linenum()

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

3.3.4　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 8　end_scan()

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

3.3.5　isNumber 関数

■機能　fgetc で収得した文字が数字 (0〜9) 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　文字が数字の場合は 1 を返し、文字が数字以外のときは 0 を返す。

ソースコード 9　isNumber()

```
1 int isNumber(int c);
```

3.3.6　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 10　isChar()

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

3.3.7　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 11　UntilFun()

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

3.3.8　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 12　UntilComment()

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

3.3.9　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 13　UntilString()

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

3.3.10　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 14　init_idtab()

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

3.3.11　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 15　id_countup()

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

3.3.12　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 16　print_idtab()

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

3.3.13　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 17　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。
テストファイルは以下の９個である。

- sample11pp.mpl
- test1.mpl
- test2.mpl
- test3.mpl
- test4.mpl
- test00.mpl
- test01.mpl
- test02.mpl
- test03.mpl
- test04.mpl
- test05.mpl
- test06.mpl
- test07.mpl

ソースコード 18　sample11pp.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.
```

ソースコード 19　test00.mpl

```
1 /*aaa*/
2 {bbb}
3 'ccc'
```

ソースコード 20　test00.mpl

```
1 /*aaa*/
2 {bbb}
3 'ccc'
```

ソースコード 21　test01.mpl

```
1 /*aaaa/
```

ソースコード 22　test02.mpl

```
1 /*aaaa
```

ソースコード 23　test03.mpl

```
1 {aaa
```

ソースコード 24　test04.mpl

```
1 'aaaa
```

ソースコード 25　test05.mpl

```
1 あ
```

ソースコード 26　test06.mpl

```
1 /
```

ソースコード 27　test06.mpl

```
1 1あ
```

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を実験室で実行した。

ソースコード 28　演習室環境での program の実行例

```
$ gcc −o program token−list.c token−list.h id−list.c scan−list.c
$ ./program sample11pp.mpl
```

結果は以下のようになった。

ソースコード 29　result

```
"NAME " 27
"program " 1
"var " 4
"begin " 5
"end " 5
"procedure " 3
"call " 3
"while " 1
"do " 1
"integer " 6
"readln " 2
"writeln " 2
"NUMBER " 4
"STRING " 2
"+ " 1
"- " 1
"* " 1
"= " 3
"> " 1
"( " 8
") " 7
":= " 9
". " 1
", " 2
"; " 17
        "Identifier" "call " 3
        "Identifier" "do " 1
        "Identifier" "while " 1
        "Identifier" "s " 4
        "Identifier" "goukei " 2
        "Identifier" "data " 4
        "Identifier" "wakakidasi " 2
        "Identifier" "sum " 3
        "Identifier" "var " 4
        "Identifier" "end " 5
```

```
36          "Identifier" "readln " 2
37          "Identifier" "writeln " 2
38          "Identifier" "begin " 5
39          "Identifier" "integer " 6
40          "Identifier" "n " 9
41          "Identifier" "kazuyomikomi " 2
42          "Identifier" "procedure " 3
43          "Identifier" "sample11pp " 1
44          "Identifier" "program " 1
```

test1.mpl には a を 1024 個並べたものを書いた。test2.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 30　a を 1024 個書いたときの結果

```
1   "NAME " 1
```

ソースコード 31　a を 1025 個書いたときの結果

```
1   ERROR(0): string is too long
```

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 32　2,147,483,647 書いたときの結果

```
1   "NAME " 1
```

ソースコード 33　2,147,483,648 書いたときの結果

```
1   ERROR(0): string is too long
```

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 34　comand.sh

```
#!/bin/bash
echo "rm *.gcda *.gcno *.gcov"
rm *.gcda *.gcno *.gcov
echo "gcc --coverage -o tc id-list.c scan-list.c token-list.c token-list.h"
gcc --coverage -o tc id-list.c scan-list.c token-list.c token-list.h
echo "./tc sample11pp.mpl"
./tc sample11pp.mpl
echo "./tc test000.mpl"
./tc test000.mpl
echo "./tc test00.mpl"
./tc test00.mpl
echo "./tc test01.mpl"
./tc test01.mpl
echo "./tc test02.mpl"
```

```
./tc test02.mpl
echo "./tc test03.mpl"
./tc test03.mpl
echo "./tc test04.mpl"
./tc test04.mpl
echo "./tc test05.mpl"
./tc test05.mpl
echo "./tc test06.mpl"
```

ソースコード 35　bash を用いた comand の実行例

```
$ bash comand.sh
```

ソースコード 36　result

```
# bash comand.sh
rm *.gcda *.gcno *.gcov
gcc --coverage -o tc id-list.c scan-list.c token-list.c token-list.h
./tc sample11pp.mpl
"NAME " 27
"program " 1
"var " 4
"begin " 5
"end " 5
"procedure " 3
"call " 3
"while " 1
"do " 1
"integer " 6
"readln " 2
"writeln " 2
"NUMBER " 4
"STRING " 2
"+ " 1
"- " 1
"* " 1
"= " 3
"> " 1
"( " 8
") " 7
":= " 9
". " 1
", " 2
"; " 17
        "Identifier" "call " 3
        "Identifier" "do " 1
        "Identifier" "while " 1
        "Identifier" "s " 4
```

```
        "Identifier" "goukei " 2
        "Identifier" "data " 4
        "Identifier" "wakakidasi " 2
        "Identifier" "sum " 3
        "Identifier" "var " 4
        "Identifier" "end " 5
        "Identifier" "readln " 2
        "Identifier" "writeln " 2
        "Identifier" "begin " 5
        "Identifier" "integer " 6
        "Identifier" "n " 9
        "Identifier" "kazuyomikomi " 2
        "Identifier" "procedure " 3
        "Identifier" "sample11pp " 1
        "Identifier" "program " 1
./tc test000.mpl
File test000.mpl can not open.
./tc test00.mpl
"STRING " 1
./tc test01.mpl

ERROR(0): /* is undeclared.another */ is expected.

./tc test02.mpl

ERROR(0): /* is undeclared.another */ is expected.

./tc test03.mpl

ERROR(0): { is undeclared. } is expected.

./tc test04.mpl

ERROR(0): ' is undeclared.another ' is expected.

./tc test05.mpl

ERROR(0): ^^ef^^bf^^bd is undeclared.

./tc test06.mpl

ERROR(0): / is not undeclared

./tc test07.mpl

ERROR(0): ^^ef^^bf^^bd is undeclared.

gcov -b scan-list.gcda
File 'scan-list.c'
```

```
83  Lines executed:95.58% of 113
84  Branches executed:100.00% of 62
85  Taken at least once:95.16% of 62
86  Calls executed:100.00% of 32
87  Creating 'scan-list.c.gcov'
```

4.3　テストデータの十分性

ここではそれだけのテストでどの程度バグがないことが保証できるかを記述する
まず、コード 29 の結果をみたとき、トークンの種類とトークンの数をプログラム (コード 28) のトークンの種類とトークンの数が一致していた。また、文字列の境界部分 1024 と 1025 の間で命令を実行すると、最大値以上の範囲ではエラーが出ることが分かった。また、数字の最大値 2147483647 を書いたコード test3.mpl と 最大値を 1 超えた 2147483647 を書いた test4.mpl で書いたとき最大値以上でエラーがでる事が分かった。

続いて、コード 34 を実行したときの結果 (コード 36) みたとき、命令網羅は 95% であった。分岐数は 62 個存在し、分岐を通過した数は 100% であった。分岐網羅率は 95.16% であった。以上のブラックボックス、ホワイトボックスのテストよりバグのないプログラムが書けたと考えられる。

5　事前計画と実際の進捗状況

5.1　事前計画

事前計画は 1 のようになった。

表 1　事前作業計画

開始予定日	終了予定日	見積もり時間	番号	作業内容
10/03	10/03	1	(a)	スケジュールを立てる
10/04	10/04	0.5	(b-1)	配布された資料を読み直す
10/04	10/04	0.5	(b-2)	配布されたプログラムを読む
10/04	10/04	1	(b-3)	コンパイラのテキスト (プログラム) を読む
10/05	10/07	5	(c)	字句解析系 (スキャナ) の概略設計
10/08	10/08	2	(e-1-1)	ブラックボックステスト用プログラムの作成
10/09	10/11	5	(d-4)	スキャナの作成
10/12	10/12	1	(e-1-2)	バグがない場合の想定テスト結果の準備
10/13	10/13	0.5	(d-1)	トークンカウント用の配列を初期化部分の作成
10/13	10/13	0.5	(d-2)	トークンをカウント部分の作成
10/13	10/13	1	(d-3)	カウントした結果の出力部分の作成
10/14	10/14	0.5	(e-2-1)	カバレッジレベルの決定
10/14	10/14	2	(e-2-2)	ホワイトボックステスト用プログラムの作成
10/15	10/15	1	(e-2-3)	バグがない場合の想定テスト結果の準備
10/16	10/20	8	(f)	テストとデバッグを行う
10/28	10/28	1	(g-1)	作成したプログラムの設計情報を書く
10/29	10/29	1	(g-2)	テスト情報を書く
10/30	10/30	1	(g-3)	事前計画と実際の進捗状況を書く
10/31	10/31	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

課題 1 の為省略。

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
10/03	10/03	1	(a)	10/03	0.5
10/04	10/04	0.5	(b-1)	10/10	1
10/04	10/04	0.5	(b-2)	10/10	1
10/04	10/04	1	(b-3)	10/10	1
10/05	10/07	5	(c)	10/10	3
10/08	10/08	2	(e-1-1)	10/10	1
10/09	10/11	5	(d-4)	10/16	1
10/12	10/12	1	(e-1-2)	10/16	5
10/13	10/13	0.5	(d-1)	10/16	1
10/13	10/13	0.5	(d-2)	10/16	1
10/13	10/13	1	(d-3)	10/16	1
10/14	10/14	0.5	(e-2-1)	10/16	1
10/14	10/14	2	(e-2-2)	10/16	1
10/15	10/15	1	(e-2-3)	10/16	1
10/16	10/20	8	(f)	10/17	5
10/28	10/28	1	(g-1)	10/18	1
10/29	10/29	1	(g-2)	10/18	1
10/30	10/30	1	(g-3)	10/18	1
10/31	10/31	5	(h)	10/31	10

5.4　当初の事前計画と実際の進捗との差の原因

表2より若干の進行に差があった。特に初期の実装が事前計画よりタイミングが遅くなった。今回は１回目ともあり、例を参考に事前計画を立てた。個人の用事を考慮できていなかったことが原因であった。また、レポートの各時間も短く見積もっていたたことが原因で直前の提出となった。

6　ソースコード

ソースコード 37　token-list.c

```
#include "token-list.h"

/* keyword list */
struct KEY key[KEYWORDSIZE] = {
        {"and",  TAND },
        {"array", TARRAY },
        {"begin", TBEGIN },
        {"boolean", TBOOLEAN},
        {"break", TBREAK },
        {"call", TCALL },
        {"char", TCHAR },
        {"div",  TDIV },
        {"do",  TDO },
        {"else", TELSE },
        {"end",  TEND },
        {"false", TFALSE },
        {"if",  TIF },
        {"integer", TINTEGER},
        {"not",  TNOT },
        {"of",  TOF },
        {"or",  TOR },
        {"procedure", TPROCEDURE},
        {"program", TPROGRAM},
        {"read", TREAD },
        {"readln", TREADLN },
        {"return",  TRETURN },
        {"then", TTHEN },
        {"true", TTRUE },
        {"var",  TVAR },
        {"while", TWHILE },
        {"write", TWRITE },
        {"writeln", TWRITELN}
};

/* Token counter */
int numtoken[NUMOFTOKEN+1] = {0};

/* string of each token */
char *tokenstr[NUMOFTOKEN+1] = {
        "",
        "NAME", "program", "var", "array", "of", "begin", "end", "if", "then",
         "else", "procedure", "return", "call", "while", "do", "not", "or",
        "div", "and", "char", "integer", "boolean", "readln", "writeln", "true",
         "false", "NUMBER", "STRING", "+", "-", "*", "=", "<>", "<", "<=", ">",
```

```
45          ">=", "(", ")", "[", "]", ":=", ".", ",", ":", ";", "read","write", "break"
46  };
47
48  int main(int nc, char *np[]) {
49          int token, i;
50
51      if(nc < 2) {
52          printf("File name id not given.\n");
53          return 0;
54      }
55      if(init_scan(np[1]) < 0) {
56          printf("File %s can not open.\n", np[1]);
57          return 0;
58      }
59
60      init_idtab() ;
61
62      while((token = scan()) >= 0) {
63          numtoken[token]++;
64      }
65      end_scan();
66
67      for(i = 0; i < NUMOFTOKEN+1;i++){
68          if(numtoken[i]>0){
69                  if((TPLUS <= i) &&(i <= TSEMI)){
70                          printf("\"%-15s \"\t%2d\n",tokenstr[i], numtoken[i]);
71                  }else{
72                  printf("\"%-15s \"\t%2d\n",tokenstr[i], numtoken[i]);
73                  }
74                  numtoken[i] = 0;
75          }
76      }
77      print_idtab();
78
79      release_idtab();
80      return 0;
81  }
82
83  void error(char *mes) {
84          int line = get_linenum();
85          printf("\nERROR(%d): %s\n", line, mes);
86          end_scan();
87          release_idtab();
88          exit(1);
89  }
```

ソースコード 38 token-list.h

```
/* token-list.h */
#include <stdlib.h>
#include <stdio.h>
#include <string.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         char * keyword;
68         int keytoken;
69 } key[KEYWORDSIZE];
70
71 extern void error(char *mes);
72 extern char* tokenstr[NUMOFTOKEN+1];
73 extern void init_idtab() ;
74 extern void id_countup(char *np);
75 extern void print_idtab();
76 extern void release_idtab();
77
78 /* list-scan.c */
79 extern int init_scan(char *filename);
80 extern int scan(void);
81 extern int num_attr;
82 extern char string_attr[MAXSTRSIZE];
83 extern int get_linenum(void);
84 extern void end_scan(void);
85 extern int isNumber(int c);
86 extern int isChar(int c);
87 extern void UntilFun(int c);
88 extern void UntilComment(void);
89 extern void UntilString(void);
90
91
92 /* token-list.c */
93 extern void init_idtab();
94 extern void id_countup(char *np);
95 extern void print_idtab();
96 extern void release_idtab();
```

ソースコード 39 scan-list.c

```
#include "token-list.h"
#define ERROR (-1)
#define EOFCODE (-2)
int cbuf, num_line = 0;
int num_attr;
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);
                //  if(cbuf == '\r'){
                //   cbuf = fgetc(fp);
                //  }
                //  num_line++;
                //  break;
                case '\n': //end of line Unix,Mac
                        cbuf = fgetc(fp);
                        num_line++;
                        break;
                case ' ': //space
                case '\t': //tab
                        cbuf = fgetc(fp);
                        break;
                case '{': //coment {...}
                        UntilFun('}');
                        break;
                case '/':
```

```
48                      cbuf = fgetc(fp);
49                      if (cbuf == '*') {
50                              UntilComment();
51                              break;
52                      }else{
53                              error("/ is not undeclared\n");
54                              return ERROR;
55                      }
56                      break;
57              case '\'':
58                      UntilString();
59                      return TSTRING;
60                      break;
61              default:
62                      isSeparator = 0;
63                      break;
64              }
65      }
66      int count = 0;
67      if (isChar(cbuf)) { // if char, read them by end
68              int i, j = 0;
69              for (i = 0; (isChar(cbuf) + isNumber(cbuf)) >= 1; i++) {
70                      string_attr[i] = cbuf;
71                      if(i >= MAXSTRSIZE){
72                              error("string is too long\n");
73                              return ERROR;
74                      }
75                      cbuf = fgetc(fp);
76                      count++;
77              }
78              //printf("string_attr: %s\n", string_attr);
79              id_countup(string_attr);
80              for (j = 0; j <= NUMOFTOKEN; j++) { //check whether keyword or name
81                      if (strcmp(string_attr, key[j].keyword) == 0) {
82                              while (count >= 0) {//init string_attr
83                                      string_attr[count] = '\0';
84                                      count--;
85                              }
86                              return key[j].keytoken; //retrun keyword
87                              break;
88                      }
89              }
90              while (count >= 0) { //init string_attr
91                      string_attr[count] = '\0';
92                      count--;
93              }
94              return TNAME; //return name
95      } else if (isNumber(cbuf)) { //if number, read them by end
96              int i = 0;
```

```
97              num_attr = 0;
98              for (i = 0; isNumber(cbuf)>0; i++) {
99                      num_attr = num_attr * 10 + (cbuf-48);
100                     cbuf = fgetc(fp);
101                     if((num_attr > MAXINTSIZE)||(num_attr < 0)){
102                             error("number is too long\n");
103                             return ERROR;
104                     }
105             }
106             return TNUMBER;
107     } else {
108             int i = TPLUS;
109             for (i = TPLUS; i <= TSEMI; i++) {
110                     if (cbuf == tokenstr[i][0]) {
111                             cbuf = fgetc(fp);
112                             return i;
113                     }
114             }
115             if(cbuf == EOF){
116                     return EOFCODE;
117             }
118
119             char dst[100];
120             snprintf(dst, sizeof dst, "%c is undeclared.\n", cbuf);
121             error(dst);
122             return ERROR;
123     }
124 }
125
126 int isNumber(int c) {
127     if (c >= '0' && c <= '9') {
128             return 1;
129     } else {
130             return 0;
131     }
132 }
133
134 int isChar(int c) {
135     if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
136             return 1;
137     } else {
138             return 0;
139     }
140 }
141
142 void UntilFun(int c) {
143     char cin = cbuf;
144     cbuf = fgetc(fp);
145     while (cbuf != c) {
```

```
146             if (cbuf == EOF) {
147                     char mes[100];
148                     sprintf(mes, "%c is undeclared. %c is expected.\n",cin,c);
149                     error(mes);
150                     break;
151             }
152             cbuf = fgetc(fp);
153     }
154     cbuf = fgetc(fp);
155 }
156
157 void UntilComment(void) {
158     while (1) {
159             cbuf = fgetc(fp);
160             if (cbuf == '*') {
161                     cbuf = fgetc(fp);
162                     if (cbuf == '/') {
163                             cbuf = fgetc(fp);
164                             break;
165                     }
166             }
167             if (cbuf == EOF) {
168                     error("/* is undeclared.another */ is expected.\n");
169                     break;
170             }
171     }
172 }
173
174 void UntilString(void) {
175     while (1) {
176             cbuf = fgetc(fp);
177             if (cbuf == '\'') {
178                     cbuf = fgetc(fp);
179                     if (cbuf != '\'') {
180                             cbuf = fgetc(fp);
181                             break;
182                     }
183             }
184             if (cbuf == EOF) {
185                     error("\' is undeclared.another \' is expected.\n");
186                     break;
187             }
188     }
189 }
190
191 int get_linenum(void) {
192     return num_line;
193 }
194 void end_scan(void) {
```

```
195         fclose(fp);
196 }
```

ソースコード 40　id-list.c

```
#include "token-list.h"

struct ID {
        char *name;
        int count;
        struct ID *nextp;
} *idroot;

void init_idtab() {  /* Initialise the table */
        idroot = NULL;
}

struct ID *search_idtab(char *np) { /* search the name pointed by np */
        struct ID *p;

        for(p = idroot; p != NULL; p = p->nextp) {
                if(strcmp(np, p->name) == 0) return(p);
        }
        return(NULL);
}

void id_countup(char *np) { /* Register and count up the name pointed by np */
        struct ID *p;
        char *cp;

        if((p = search_idtab(np)) != NULL) p->count++;
        else {
                if((p = (struct ID *)malloc(sizeof(struct ID))) == NULL) {
                        printf("can not malloc in id_countup\n");
                        return;
                }
                if((cp = (char *)malloc(strlen(np)+1)) == NULL) {
                        printf("can not malloc-2 in id_countup\n");
                        return;
                }
                strcpy(cp, np);
                p->name = cp;
                p->count = 1;
                p->nextp = idroot;
                idroot = p;
        }
}

void print_idtab() { /* Output the registered data */
        struct ID *p;

        for(p = idroot; p != NULL; p = p->nextp) {
```

```
48                  if(p->count != 0)
49                          printf("\t\"Identifier\" \"%-15s\"\t%2d\n", p->name, p->count);
50          }
51  }
52
53  void release_idtab() { /* Release tha data structure */
54          struct ID *p, *q;
55
56          for(p = idroot; p != NULL; p = q) {
57                  free(p->name);
58                  q = p->nextp;
59                  free(p);
60          }
61          init_idtab();
62  }
```

7 参考文献

参考文献

[1] https://www.techscore.com/blog/2016/10/05/