

# 言語処理プログラミング

情報工学 3 年 20122041 中田継太

2022 年 10 月 20 日

## 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 . . . . .	4
4	テスト情報	5
4.1	テストデータ . . . . .	5
4.2	テスト結果 . . . . .	5
4.3	テストデータの十分性 . . . . .	5
5	事前計画と実際の進捗状況	5
5.1	事前計画 . . . . .	5
5.2	事前計画の立て方についての前課題からの改善点 . . . . .	6
5.3	実際の進捗状況 . . . . .	6
5.4	当初の事前計画と実際の進捗との差の原因 . . . . .	7
6	ソースコード	8
7	参考文献	18

## 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[KEYWORDSIZ];
```

### ■変数

cub 入力には常に 1 文字先読みしておく と便利である。即ち、1 文字分の文字バッファを持っていて、それに次の文字が入っているようにする。以降、この文字バッファを

ソースコード 3 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。

## 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。

### 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。

図 1 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 test.txt
```

### 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

## 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	-	(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	-	(h)		

#### 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。

## 6 ソースコード

ソースコード 4 token-list.c

```
1 #include "token-list.h"
2
3 /* keyword list */
4 struct KEY key[KEYWORDSIZE] = {
5     {"and", TAND },
6     {"array", TARRAY },
7     {"begin", TBEGIN },
8     {"boolean", TBOOLEAN},
9     {"break", TBREAK },
10    {"call", TCALL },
11    {"char", TCHAR },
12    {"div", TDIV },
13    {"do", TDO },
14    {"else", TELSE },
15    {"end", TEND },
16    {"false", TFALSE },
17    {"if", TIF },
18    {"integer", TINTEGER},
19    {"not", TNOT },
20    {"of", TOF },
21    {"or", TOR },
22    {"procedure", TPROCEDURE},
23    {"program", TPROGRAM},
24    {"read", TREAD },
25    {"readln", TREADLN },
26    {"return", TRETURN },
27    {"then", TTHEN },
28    {"true", TTRUE },
29    {"var", TVAR },
30    {"while", TWHILE },
31    {"write", TWRITE },
32    {"writeln", TWRITELN}
33 };
34
35 /* Token counter */
36 int numtoken[NUMOFTOKEN+1] = {0};
37
38 /* string of each token */
39 char *tokenstr[NUMOFTOKEN+1] = {
40     "",
41     "NAME", "program", "var", "array", "of", "begin", "end", "if", "then",
42     "else", "procedure", "return", "call", "while", "do", "not", "or",
43     "div", "and", "char", "integer", "boolean", "readln", "writeln", "true",
44     "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     printf("\n ERROR: %s\n", mes);
85     end_scan();
86 }
```

ソースコード 5 token-list.h

```
1  /* token-list.h */
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5
6  #define MAXSTRSIZE 1024
7
8  /* Token */
9  #define TNAME 1 /* Name : Alphabet { Alphabet | Digit } */
10 #define TPROGRAM 2 /* program : Keyword */
11 #define TVAR 3 /* var : Keyword */
12 #define TARRAY 4 /* array : Keyword */
13 #define TOF 5 /* of : Keyword */
14 #define TBEGIN 6 /* begin : Keyword */
15 #define TEND 7 /* end : Keyword */
16 #define TIF 8 /* if : Keyword */
17 #define TTHEN 9 /* then : Keyword */
18 #define TELSE 10 /* else : Keyword */
19 #define TPROCEDURE 11 /* procedure : Keyword */
20 #define TRETURN 12 /* return : Keyword */
21 #define TCALL 13 /* call : Keyword */
22 #define TWHILE 14 /* while : Keyword */
23 #define TDO 15 /* do : Keyword */
24 #define TNOT 16 /* not : Keyword */
25 #define TOR 17 /* or : Keyword */
26 #define TDIV 18 /* div : Keyword */
27 #define TAND 19 /* and : Keyword */
28 #define TCHAR 20 /* char : Keyword */
29 #define TINTEGER 21 /* integer : Keyword */
30 #define TBOOLEAN 22 /* boolean : Keyword */
31 #define TREADLN 23 /* readln : Keyword */
32 #define TWRITELN 24 /* writeln : Keyword */
33 #define TTRUE 25 /* true : Keyword */
34 #define TFALSE 26 /* false : Keyword */
35 #define TNUMBER 27 /* unsigned integer */
36 #define TSTRING 28 /* String */
37 #define TPLUS 29 /* + : symbol */
38 #define TMINUS 30 /* - : symbol */
39 #define TSTAR 31 /* * : symbol */
40 #define TEQUAL 32 /* = : symbol */
41 #define TNOTEQ 33 /* <> : symbol */
42 #define TLE 34 /* < : symbol */
43 #define TLEEQ 35 /* <= : symbol */
44 #define TGR 36 /* > : symbol */
45 #define TGREQ 37 /* >= : symbol */
46 #define TLPAREN 38 /* ( : symbol */
47 #define TRPAREN 39 /* ) : symbol */
```

```
48 #define TLSQPAREN 40 /* [ : symbol */
49 #define TRSQPAREN 41 /* ] : symbol */
50 #define TASSIGN 42 /* := : symbol */
51 #define TDOT 43 /* . : symbol */
52 #define TCOMMA 44 /* , : symbol */
53 #define TCOLON 45 /* : : symbol */
54 #define TSEMI 46 /* ; : symbol */
55 #define TREAD 47 /* read : Keyword */
56 #define TWRITE 48 /* write : Keyword */
57 #define TBREAK 49 /* break : Keyword */
58
59 #define NUMOFTOKEN 49
60
61 /* token-list.c */
62
63 #define KEYWORDSIZE 28
64
65 extern struct KEY {
66     char * keyword;
67     int keytoken;
68 } key[KEYWORDSIZE];
69
70 extern void error(char *mes);
71 extern char* tokenstr[NUMOFTOKEN+1];
72
73 extern void init_idtab() ;
74 extern void id_countup(char *np);
75 extern void print_idtab();
76 extern void release_idtab();
77
78 /* 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);
```

ソースコード 6 scan-list.c

```
1 #include "token-list.h"
2 int cbuf, num_line = 0;
3 int num_attr;
4 char string_attr[MAXSTRSIZE];
5 FILE *fp = NULL;
6
7 int init_scan(char *filename) { /* open file if it succeed return 0 and if not return -1 */
8     if ((fp = fopen(filename, "r+")) == NULL) {
9         printf("cannot open filename");
10        return -1;
11    } else {
12        cbuf = fgetc(fp);
13        return 0;
14    }
15 }
16
17 int scan(void) {
18     int isSeparator = 1;
19     while (isSeparator) { // if separator, skip read
20         switch (cbuf) {
21             case '\r': //end of line
22                 cbuf = fgetc(fp);
23                 if (cbuf == '\n') {
24                     cbuf = fgetc(fp);
25                 }
26                 num_line++;
27                 break;
28             case '\n': //end of line
29                 cbuf = fgetc(fp);
30                 if (cbuf == '\r') {
31                     cbuf = fgetc(fp);
32                 }
33                 num_line++;
34                 break;
35             case ' ': //space
36             case '\t': //tab
37                 cbuf = fgetc(fp);
38                 break;
39             case '{': //coment {...}
40                 UntilFun('}');
41                 break;
42             case '/':
43                 UntilComment();
44                 break;
45             case '\\':
46                 UntilFun('\\');
47                 return TSTRING;
```

```

48             break;
49         default:
50             isSeparator = 0;
51             break;
52     }
53 }
54 int count = 0;
55 if (isChar(cbuf)) { // if char, read them by end
56     int i, j = 0;
57     for (i = 0; (isChar(cbuf) + isNumber(cbuf)) >= 1; i++) {
58         string_attr[i] = cbuf;
59         cbuf = fgetc(fp);
60         count++;
61     }
62     //printf("string_attr: %s\n", string_attr);
63     id_countup(string_attr);
64     for (j = 0; j <= NUMOFTOKEN; j++) { //check whether keyword or name
65         if (strcmp(string_attr, key[j].keyword) == 0) {
66             while (count >= 0) {
67                 string_attr[count] = '\0';
68                 count--;
69             }
70             return key[j].keytoken; //return keyword
71             break;
72         }
73     }
74     while (count >= 0) {
75         string_attr[count] = '\0';
76         count--;
77     }
78     return TNAME; //return name
79 } else if (isNumber(cbuf)) { //if number, read them by end
80     int i = 0;
81     num_attr = 0;
82     for (i = 0; isNumber(cbuf); i++) {
83         num_attr = num_attr * 10 + cbuf;
84         cbuf = fgetc(fp);
85     }
86     return TNUMBER;
87 } else {
88     int i = TPLUS;
89     for (i = TPLUS; i <= TSEMI; i++) {
90         if (cbuf == tokenstr[i][0]) {
91             cbuf = fgetc(fp);
92             return i;
93         }
94     }
95 }
96 return -1;

```

```
97 }
98
99 int isNumber(int c) {
100     if (c >= '0' && c <= '9') {
101         return 1;
102     } else {
103         return 0;
104     }
105 }
106
107 int isChar(int c) {
108     if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
109         return 1;
110     } else {
111         return 0;
112     }
113 }
114
115 void UntilFun(int c) {
116     cbuf = fgetc(fp);
117     while (cbuf != c) {
118         if (c == EOF) {
119             break;
120         }
121         cbuf = fgetc(fp);
122     }
123     cbuf = fgetc(fp);
124 }
125
126 void UntilComment(void) {
127     cbuf = fgetc(fp);
128     if (cbuf == '/*') {
129         while (1) {
130             cbuf = fgetc(fp);
131             if (cbuf == '*/') {
132                 cbuf = fgetc(fp);
133                 if (cbuf == '/') {
134                     cbuf = fgetc(fp);
135                     break;
136                 }
137             }
138             if (cbuf == EOF) {
139                 break;
140             }
141         }
142     }
143 }
144
145 int get_linenum(void) {
```

```
146     return num_line;
147 }
148 void end_scan(void) {
149     fclose(fp);
150 }
```

ソースコード 7 id-list.c

```
1 #include "token-list.h"
2
3 struct ID {
4     char *name;
5     int count;
6     struct ID *nextp;
7 } *idroot;
8
9 void init_idtab() { /* Initialise the table */
10     idroot = NULL;
11 }
12
13 struct ID *search_idtab(char *np) { /* search the name pointed by np */
14     struct ID *p;
15
16     for(p = idroot; p != NULL; p = p->nextp) {
17         if(strcmp(np, p->name) == 0) return(p);
18     }
19     return(NULL);
20 }
21
22 void id_countup(char *np) { /* Register and count up the name pointed by np */
23     struct ID *p;
24     char *cp;
25
26     if((p = search_idtab(np)) != NULL) p->count++;
27     else {
28         if((p = (struct ID *)malloc(sizeof(struct ID))) == NULL) {
29             printf("can not malloc in id_countup\n");
30             return;
31         }
32         if((cp = (char *)malloc(strlen(np)+1)) == NULL) {
33             printf("can not malloc-2 in id_countup\n");
34             return;
35         }
36         strcpy(cp, np);
37         p->name = cp;
38         p->count = 1;
39         p->nextp = idroot;
40         idroot = p;
41     }
42 }
43
44 void print_idtab() { /* Output the registered data */
45     struct ID *p;
46
47     for(p = idroot; p != NULL; p = p->nextp) {
```



```
48         if(p->count != 0)
49             printf("\t\tIdentifier\ " "%-15s\ "\t%2d\n", p->name, p->count);
50     }
51 }
52
53 void release_idtab() { /* Release the 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/>