

実験レポート 1

坪井正太郎 (101830245)

2020 年 12 月 17 日

1 実験課題 1

1.1 課題概要

scanner.l のルール部を更新し、lex によって字句解析のための C コードを生成する。

1.2 実装

scanner.l に、以下のようなルールを追加した。

ソースコード 1 scanner.l(ルールの記述)

```
1  begin return SBEGIN;
2  div return DIV;
3
4  do return DO;
5  else return ELSE;
6  end return SEND;
7  for return FOR;
8  forward return FORWARD;
9  function return FUNCTION;
10 if return IF;
11 procedure return PROCEDURE;
12 program return PROGRAM;
13 read return READ;
14 then return THEN;
15 to return TO;
16 var return VAR;
17
18 while return WHILE;
19 write return WRITE;
20
21 "+" return PLUS;
22 "-" return MINUS;
23
24 "*" return MULT;
```

```

25  "=" return EQ;
26  "<>" return NEQ;
27  "<" return LT;
28  "<=" return LE;
29  ">" return GT;
30  ">=" return GE;
31  "(" return LPAREN;
32  ")" return RPAREN;
33  "[" return LBRACKET;
34  "]" return RBRACKET;
35  "," return COMMA;
36  ";" return SEMICOLON;
37  ":" return COLON;
38  ".." return INTERVAL;
39
40  "." return PERIOD;
41  ":=" return ASSIGN;

```

1.3 実行結果

\$. /scanner ex1.p を実行した結果、以下のような出力となった。

ソースコード 2 ./scanner ex1.p の実行結果

```

1  "program": 11 RESERVE
2  "EX1": 38 EX1
3  ";": 32 RESERVE
4  "var": 15 RESERVE
5  "x": 38 x
6  ",": 31 RESERVE
7  "y": 38 y
8  ",": 31 RESERVE
9  "z": 38 z
10 ";": 32 RESERVE
11 "begin": 1 RESERVE
12 "x": 38 x
13 " := ": 36 RESERVE
14 "12": 37 12
15 ";": 32 RESERVE
16 "y": 38 y
17 " := ": 36 RESERVE
18 "20": 37 20
19 ";": 32 RESERVE
20 "z": 38 z
21 " := ": 36 RESERVE
22 "x": 38 x

```

```
23  "+" : 18 RESERVE
24  "y" : 38 y
25  "end" : 5 RESERVE
26  "." : 35 RESERVE
```

1.4 考察

scanner.l では予約語に対して、symbols.h で定義されている列挙体の、対応する要素を返している。実際に実験結果では、定義されたものに対応する整数と、予約語であれば RESERVE、識別子であればそれが表示される。

2 実験課題 2

2.1 課題概要

PL-0 を受理するような構文解析プログラムを実装する。

PL-0 の構文規則を参考に、paser.y を編集して yacc によって y.tab.h と y.tab.c を生成する。実際に解析するためには、scanner.l の中で構文解析部を呼び出すための編集も行う。

2.2 実装

構文の定義に従い、parser.y には構文要素とその定義を追加した。また、エラー時には字句解析部で用意される行番号とエラー時の字句を出力するよう編集を行った。

ソースコード 3 parser.y

```
1  %{
2  /*
3   * parser; Parser for PL-0
4   */
5
6  #define MAXLENGTH 16
7
8  #include <stdio.h>
9
10 int yyparse();
11 int yyerror(char *);
12
13 extern int yylineno;
14 extern char *yytext;
15
16 %}
17
18 %union {
19     int num;
```

```

20     char ident[MAXLENGTH+1];
21 }
22
23 %token SBEGIN DO ELSE SEND
24 %token FOR FORWARD FUNCTION IF PROCEDURE
25 %token PROGRAM READ THEN TO VAR
26 %token WHILE WRITE
27
28 %left PLUS MINUS
29 %left MULT DIV
30
31 %token EQ NEQ LE LT GE GT
32 %token LPAREN RPAREN LBRACKET RBRACKET
33 %token COMMA SEMICOLON COLON INTERVAL
34 %token PERIOD ASSIGN
35 %token <num> NUMBER
36 %token <ident> IDENT
37
38 %%
39
40 program
41     : PROGRAM IDENT SEMICOLON outblock PERIOD
42     ;
43
44 outblock
45     : var_decl_part subprog_decl_part statement
46     ;
47
48 var_decl_part
49     : /* empty */
50     | var_decl_list SEMICOLON
51     ;
52 var_decl_list
53     : var_decl_list SEMICOLON var_decl
54     | var_decl
55     ;
56
57 var_decl
58     : VAR id_list
59     ;
60
61 subprog_decl_part
62     : /* empty */
63     | subprog_decl_list SEMICOLON
64     ;
65

```

```

66 subprog_decl_list
67     : subprog_decl
68     | subprog_decl_list SEMICOLON subprog_decl
69     ;
70
71 subprog_decl
72     : proc_decl
73     ;
74
75 proc_decl
76     : PROCEDURE proc_name SEMICOLON inblock
77     ;
78
79 proc_name
80     : IDENT
81     ;
82
83 inblock
84     : var_decl_part statement
85     ;
86
87 statement_list
88     : statement
89     | statement_list SEMICOLON statement
90     ;
91
92 statement
93     : assignment_statement
94     | if_statement
95     | while_statement
96     | for_statement
97     | proc_call_statement
98     | null_statement
99     | block_statement
100    | read_statement
101    | write_statement
102    ;
103
104 assignment_statement
105     : IDENT ASSIGN expression
106     ;
107
108 if_statement
109     : IF condition THEN statement else_statement
110     ;
111

```

```

112 else_statement
113     : /* empty */
114     | ELSE statement
115     ;
116
117 while_statement
118     : WHILE condition DO statement
119     ;
120
121 for_statement
122     : FOR IDENT ASSIGN expression TO expression DO statement
123     ;
124
125 proc_call_statement
126     : proc_call_name
127     ;
128
129 proc_call_name
130     : IDENT
131     ;
132
133 block_statement
134     : SBEGIN statement_list SEND
135     ;
136
137 read_statement
138     : READ LPAREN IDENT RPAREN
139     ;
140
141 write_statement
142     : WRITE LPAREN expression RPAREN
143     ;
144
145 null_statement
146     : /* empty */
147     ;
148
149 condition
150     : expression EQ expression
151     | expression NEQ expression
152     | expression GT expression
153     | expression GE expression
154     | expression LT expression
155     | expression LE expression
156     ;
157

```

```

158 expression
159     : term
160     | PLUS term
161     | MINUS term
162     | expression PLUS expression
163     | expression MINUS expression
164     ;
165
166 term
167     : factor
168     | term MULT factor
169     | term DIV factor
170     ;
171
172 factor
173     : var_name
174     | NUMBER
175     | LPAREN expression RPAREN
176     ;
177
178 var_name
179     : IDENT
180     ;
181
182 arg_list
183     : expression
184     | arg_list COMMA expression
185     ;
186
187 id_list
188     : IDENT
189     | id_list COMMA IDENT
190     ;
191
192
193 %%
194 yyerror(char *s)
195 {
196     fprintf(stderr, "%s\n", s);
197     fprintf(stderr, "%s\n", yytext);
198     fprintf(stderr, "%d\n", yylineno);
199 }

```

ソースコード 4 scanner.l

```

1  %{
2  /*

```

```

3      * scanner: scanner for PL-*
4      *
5      */
6
7      #include <stdio.h>
8      #include <string.h>
9
10     #define MAXLENGTH 16
11
12     #include "y.tab.h"
13
14     %}
15     %option yylineno
16     %%
17
18     begin return SBEGIN;
19     div return DIV;
20
21     省略
22
23     %%
24
25     main(int argc, char *argv[]) {
26         FILE *fp;
27         int tok;
28
29         if (argc != 2) {
30             fprintf(stderr, "usage: %s filename\n", argv[0]);
31             exit(1);
32         }
33
34         if ((fp = fopen(argv[1], "r")) == NULL) {
35             fprintf(stderr, "cannot open file: %s\n", argv[1]);
36             exit(1);
37         }
38
39         /*
40          * yyin は lex の内部変数であり、入力のファイルポインタを表す。
41          */
42         yyin = fp;
43
44         yyparse();
45     }

```

2.3 実行結果

正しいプログラムが渡された場合、出力はなかった。誤りのあるプログラム `pl0a-err.p` を入力した場合、出力は以下のようになった。

ソースコード 5 `pl0a-err.p` の出力

```
1 $ ./parser pl0a-err.p
2 syntax error
3 while
4 6
```

2.4 考察

`parser.y` のエラー出力部では、`yytext`、`yylineno` のグローバル変数を参照して構文解析のエラー結果を出力している。

また、実際に構文解析を行うためには、字句解析の内容を `parser` にわたす必要がある。そのために、`scanner.l` では `yacc` で生成された `y.tab.h` をインクルードして `yyparse` 関数によって構文解析部を呼び出した。

3 実験課題 3

3.1 課題概要

PL-0 の構文解析時に記号表を作成し、管理できるように `symtab.h`、`symtab.c` を編集し、`parser.y` の適切な部分で呼び出す実装を行う。

シンボルのスコープや、名前を追加、検索、削除できるようにする。

3.2 実装

線形リストによって記号表を管理するように `symtab` を編集し、各操作を `parser.y` の受理部分の途中で呼び出すような実装を行なった。

`symtab.h` では、記号表のプロパティを持つ `Row` 構造体を定義し、`Row` 型と自身へのポインタを持つ自己参照構造体として `Symtab` を定義した。各操作を行う関数としては `insert`、`lookup`、`delete` を用意した。

ソースコード 6 `symtab.h`

```
1 #ifndef __SYMBOLS_H__
2 #define __SYMBOLS_H__
3
4 /* 記号表の管理 + 変数・定数の区別用 */
5 typedef enum {
6     GLOBAL_VAR, /* 大域変数 */
7     LOCAL_VAR, /* 局所変数 */
8     PROC_NAME, /* 手続き */
9 }
```

```

9   CONSTANT /* 定数 */
10 } Scope;
11
12 /* 記号表の構造体の宣言 */
13
14 // 表の列
15 typedef struct row Row;
16 struct row {
17     char* name;
18     int regnum;
19     Scope scope;
20 };
21
22 // スタックっぽい感じで表現、FILO なので出力は下から積み上がる感じに出力される
23 typedef struct symtab Symtab;
24 struct symtab {
25     Row row;
26     struct symtab* prev;
27 };
28
29 // もろもろ初期化
30 void init();
31
32 // 先頭に挿入
33 void insert(char*, int, Scope);
34
35 // 失敗したときにNULLを返したいのでポインタにしている。あんまりよくないかも。
36 Row* lookup(char*);
37
38 // 消したシンボルの数出力
39 int delete ();
40
41 // いい感じで出力
42 void printRow(Row);
43
44 #endif

```

symtab.c では、各操作関数から操作するグローバル変数 TABLE を定義した。

ソースコード 7 symtab.c

```

1 #include "symtab.h"
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6

```

```

7 #define DEBUG
8 /* 記号表の実体をここで作成 */
9 /* parser.y からは、以下の insert/lookup/delete などを通してアクセスする */
10 Symtab* TABLE;
11
12 /* insert, lookup, delete の実装 */
13
14 void init() {
15     TABLE = NULL;
16
17 #ifdef DEBUG
18     printf("[DEBUG] new symbol table created\n");
19 #endif
20 }
21
22 void insert(char* aName, int aRegnum, Scope aScope) {
23     // TODO malloc の例外処理
24     char* tName = (char*)malloc(sizeof(char) * strlen(aName));
25     strcpy(tName, aName);
26     // TODO malloc の例外処理
27     Symtab* tTable = (Symtab*)malloc(sizeof(Symtab));
28
29     // 多重にmalloc しなくても良いように、Row は値で渡す。問題ないはず?
30     Row tRow = {tName, aRegnum, aScope};
31     tTable->row = tRow;
32     tTable->prev = TABLE;
33     TABLE = tTable;
34
35 #ifdef DEBUG
36     // デバッグ出力、全部出す
37     printf("[DEBUG] inserted\n");
38     for (Symtab* tPointer = TABLE; tPointer != NULL; tPointer = tPointer->prev) {
39         printRow(tPointer->row);
40     }
41     printf("\n");
42 #endif
43 }
44
45 Row* lookup(char* aName) {
46 #ifdef DEBUG
47     printf("[DEBUG] searching: %s\n", aName);
48 #endif
49
50     for (Symtab* tPointer = TABLE; tPointer != NULL; tPointer = tPointer->prev) {
51         if (strcmp(tPointer->row.name, aName) == 0) {
52 #ifdef DEBUG

```

```

53     // デバッグ出力、見つけたもの
54     printf("[DEBUG] found\n");
55     printRow(tPointer->row);
56     printf("\n");
57 #endif
58     return &tPointer->row;
59 }
60 }
61
62 #ifdef DEBUG
63     printf("[DEBUG] not found\n\n");
64 #endif
65     return NULL;
66 }
67
68 int delete () {
69     int tCnt = 0;
70     for (Symtab* tPointer = TABLE; tPointer != NULL;) {
71         tPointer = tPointer->prev;
72
73         if (TABLE->row.scope == GLOBAL_VAR || TABLE->row.scope == PROC_NAME) {
74             break;
75         } else {
76             free(TABLE->row.name);
77             free(TABLE);
78             TABLE = tPointer;
79             tCnt++;
80         }
81     }
82
83 #ifdef DEBUG
84     // デバッグ出力、全部出す
85     printf("[DEBUG] deleted: %d symbols\n", tCnt);
86     for (Symtab* tPointer = TABLE; tPointer != NULL; tPointer = tPointer->prev)
87         printRow(tPointer->row);
88     printf("\n");
89 #endif
90
91     return tCnt;
92 }
93
94 void printRow(Row aRow) {
95     char* tScopeTable[] = {"global", "local", "proc", "const"};
96     printf("<NAME: %s, REGNUM: %d, SCOPE: %s>\n", aRow.name, aRow.regnum,
97         tScopeTable[aRow.scope]);
98 }

```

ソースコード 8 parser.y

```
1 %{
2 /*
3  * parser; Parser for PL-0
4  */
5
6 #define MAXLENGTH 16
7
8 #include <stdio.h>
9 #include <string.h>
10 #include "syntab.h"
11
12 int yyparse();
13 int yyerror(char *);
14
15 extern int yylineno;
16 extern char *yytext;
17
18 int regnum;
19 Scope scope;
20 char names[256];
21
22 %}
23
24 %union {
25     int num;
26     char ident[MAXLENGTH+1];
27 }
28
29 %token SBEGIN DO ELSE SEND
30 %token FOR FORWARD FUNCTION IF PROCEDURE
31 %token PROGRAM READ THEN TO VAR
32 %token WHILE WRITE
33
34 %left PLUS MINUS
35 %left MULT DIV
36
37 %token EQ NEQ LE LT GE GT
38 %token LPAREN RPAREN LBRACKET RBRACKET
39 %token COMMA SEMICOLON COLON INTERVAL
40 %token PERIOD ASSIGN
41 %token <num> NUMBER
42 %token <ident> IDENT
43
```

```

44  %%
45
46  program
47      :PROGRAM IDENT SEMICOLON
48      {
49          regnum = 0;
50          init();
51          scope = GLOBAL_VAR;
52      }
53      outblock PERIOD
54      ;
55
56  outblock
57      : var_decl_part subprog_decl_part statement
58      ;
59
60  var_decl_part
61      : /* empty */
62      | var_decl_list SEMICOLON
63      ;
64
65  var_decl_list
66      : var_decl_list SEMICOLON var_decl
67      | var_decl
68      ;
69
70  var_decl
71      : VAR id_list
72      ;
73
74  subprog_decl_part
75      : /* empty */
76      | subprog_decl_list SEMICOLON
77      ;
78
79  subprog_decl_list
80      : subprog_decl
81      | subprog_decl_list SEMICOLON subprog_decl
82      ;
83
84  subprog_decl
85      : proc_decl
86      ;
87
88  proc_decl
89      : PROCEDURE proc_name SEMICOLON inblock

```

```

90      {
91          scope = GLOBAL_VAR;
92          delete();
93      }
94      ;
95
96 proc_name
97      : IDENT
98      {
99          insert($1, regnum, PROC_NAME);
100         scope = LOCAL_VAR;
101     }
102     ;
103
104 inblock
105     : var_decl_part statement
106     ;
107
108 statement_list
109     : statement
110     | statement_list SEMICOLON statement
111     ;
112
113 statement
114     : assignment_statement
115     | if_statement
116     | while_statement
117     | for_statement
118     | proc_call_statement
119     | null_statement
120     | block_statement
121     | read_statement
122     | write_statement
123     ;
124
125 assignment_statement
126     : IDENT
127     {
128         lookup($1);
129     }
130     ASSIGN expression
131     ;
132
133 if_statement
134     : IF condition THEN statement else_statement
135     ;

```

```

136
137 else_statement
138     : /* empty */
139     | ELSE statement
140     ;
141
142 while_statement
143     : WHILE condition DO statement
144     ;
145
146 for_statement
147     : FOR IDENT
148     {
149         lookup($2);
150     }
151     ASSIGN expression TO expression DO statement
152     ;
153
154 proc_call_statement
155     : proc_call_name
156     ;
157
158 proc_call_name
159     : IDENT
160     {
161         lookup($1);
162     }
163     ;
164
165 block_statement
166     : SBEGIN statement_list SEND
167     ;
168
169 read_statement
170     : READ LPAREN IDENT RPAREN
171     {
172         lookup($IDENT);
173     }
174     ;
175
176 write_statement
177     : WRITE LPAREN expression RPAREN
178     ;
179
180 null_statement
181     : /* empty */

```



```

182         ;
183
184 condition
185     : expression EQ expression
186     | expression NEQ expression
187     | expression GT expression
188     | expression GE expression
189     | expression LT expression
190     | expression LE expression
191     ;
192
193 expression
194     : term
195     | PLUS term
196     | MINUS term
197     | expression PLUS expression
198     | expression MINUS expression
199     ;
200
201 term
202     : factor
203     | term MULT factor
204     | term DIV factor
205     ;
206
207 factor
208     : var_name
209     | NUMBER
210     | LPAREN expression RPAREN
211     ;
212
213 var_name
214     : IDENT
215     {
216         lookup($1);
217     }
218     ;
219
220 arg_list
221     : expression
222     | arg_list COMMA expression
223     ;
224
225 id_list
226     : IDENT
227     {

```

```

228             insert($1, regnum, scope);
229         }
230         | id_list COMMA IDENT
231         {
232             insert($3, regnum, scope);
233         }
234         ;
235
236
237 %%
238 int yyerror(char *s)
239 {
240     fprintf(stderr, "%s\n", s);
241     fprintf(stderr, "%s\n", yytext);
242     fprintf(stderr, "%d\n", yylineno);
243     return yylineno;
244 }

```

3.3 実行結果

pl0b.p を受理した結果、記号表の遷移を示した出力は以下ようになる。エラー時には、エラー部までは受理された分の記号表を出力し、課題 2 のようなエラー表示を出力する。

ソースコード 9 ./parser pl0b.p の実行結果

```

1  [DEBUG] new symbol table created
2  [DEBUG] inserted
3  <NAME: n, REGNUM: 0, SCOPE: global>
4
5  [DEBUG] inserted
6  <NAME: x, REGNUM: 0, SCOPE: global>
7  <NAME: n, REGNUM: 0, SCOPE: global>
8
9  [DEBUG] inserted
10 <NAME: prime, REGNUM: 0, SCOPE: proc>
11 <NAME: x, REGNUM: 0, SCOPE: global>
12 <NAME: n, REGNUM: 0, SCOPE: global>
13
14 [DEBUG] inserted
15 <NAME: m, REGNUM: 0, SCOPE: local>
16 <NAME: prime, REGNUM: 0, SCOPE: proc>
17 <NAME: x, REGNUM: 0, SCOPE: global>
18 <NAME: n, REGNUM: 0, SCOPE: global>
19
20 [DEBUG] searching: m
21 [DEBUG] found

```

```

22 <NAME: m, REGNUM: 0, SCOPE: local>
23
24 [DEBUG] searching: x
25 [DEBUG] found
26 <NAME: x, REGNUM: 0, SCOPE: global>
27
28 [DEBUG] searching: x
29 [DEBUG] found
30 <NAME: x, REGNUM: 0, SCOPE: global>
31
32 [DEBUG] searching: x
33 [DEBUG] found
34 <NAME: x, REGNUM: 0, SCOPE: global>
35
36 [DEBUG] searching: m
37 [DEBUG] found
38 <NAME: m, REGNUM: 0, SCOPE: local>
39
40 [DEBUG] searching: m
41 [DEBUG] found
42 <NAME: m, REGNUM: 0, SCOPE: local>
43
44 [DEBUG] searching: m
45 [DEBUG] found
46 <NAME: m, REGNUM: 0, SCOPE: local>
47
48 [DEBUG] searching: m
49 [DEBUG] found
50 <NAME: m, REGNUM: 0, SCOPE: local>
51
52 [DEBUG] searching: m
53 [DEBUG] found
54 <NAME: m, REGNUM: 0, SCOPE: local>
55
56 [DEBUG] searching: x
57 [DEBUG] found
58 <NAME: x, REGNUM: 0, SCOPE: global>
59
60 [DEBUG] deleted: 1 symbols
61 <NAME: prime, REGNUM: 0, SCOPE: proc>
62 <NAME: x, REGNUM: 0, SCOPE: global>
63 <NAME: n, REGNUM: 0, SCOPE: global>
64
65 [DEBUG] searching: n
66 [DEBUG] found
67 <NAME: n, REGNUM: 0, SCOPE: global>

```

```
68
69 [DEBUG] searching: n
70 [DEBUG] found
71 <NAME: n, REGNUM: 0, SCOPE: global>
72
73 [DEBUG] searching: x
74 [DEBUG] found
75 <NAME: x, REGNUM: 0, SCOPE: global>
76
77 [DEBUG] searching: n
78 [DEBUG] found
79 <NAME: n, REGNUM: 0, SCOPE: global>
80
81 [DEBUG] searching: prime
82 [DEBUG] found
83 <NAME: prime, REGNUM: 0, SCOPE: proc>
84
85 [DEBUG] searching: n
86 [DEBUG] found
87 <NAME: n, REGNUM: 0, SCOPE: global>
88
89 [DEBUG] searching: n
90 [DEBUG] found
91 <NAME: n, REGNUM: 0, SCOPE: global>
```

3.4 考察

3.4.1 symtab の実装

記号表を片方向連結リストとして定義したことによって、insert 処理ではポインタの付け替えが最小限で済み、delete、lookup での検索処理では最新のものから順番に辿って行けるため、効率が良い。また、symtab.c でグローバル変数 TABLE を用意し、操作は各操作関数を通して行うことで、意図しない操作を防いだ。

実装上の工夫として、ifdef と DEBUG フラグによる制御によって、記号表の出力を切り替えることができるようにした。これによって、今後の課題での修正が用意になり、デバッグの効率も上がると考えられる。

3.4.2 parser.y の追加実装

insert、lookup はそれぞれプロセス名と変数名の定義と参照部で呼び出した。

delete は、構文要素 proc_decl の最後で呼び出し、inblock 内でのみ使われる局所変数を削除した。同時に、変数の scope をローカルからグローバルに切り替えた。