

```

1  /*
2
3  再帰的下向き構文解析プログラム
4  (以下の文法に対する構文解析プログラム)
5  Expr   -> CON
6          | FID '(' List ')'
7  List   -> Expr Rest
8  Rest   -> ',' Expr
9          | ε
10
11  -- 以下は終端記号: 字句解析部で処理
12  CON    -> '0' | '1' | ... | '9'  -- 一桁の数のみ
13  FID    -> '+' | '-' | '*' | '!'
14
15
16  -- 予測型構文解析表
17          CON          FID          (          )          ,          $
18  Expr    CON          FID '(' List ')'
19  List    Expr Rest    Expr Rest
20  Rest    ε            ',' Expr
21
22  */
23
24  #include <stdio.h>
25  #include <stdlib.h> /* exit()用 */
26  #include <ctype.h> /* isdigit()用 */
27
28  /* 大域変数の宣言 */
29  int token; /* 入力の先頭のトークンを表す */
30  int yylval; /* tokenの属性 */
31
32  int column = 0; /* デバッグ用 */
33
34  /* 終端記号に対応するマクロの定義 */
35  #define CON 256
36  #define FID 257
37
38  /* 簡易字句解析ルーチン */
39  int yylex(void) {
40      int c;
41
42      if (token == '\n') { /* 前のトークンが改行だったら */
43          column=0;
44      }
45
46      do {
47          c = getchar ();
48          column++;
49      } while (c == ' ' || c == '\t');
50
51      if (isdigit (c)) {
52          yylval = c-'0'; /* 数字から数へ変換 */
53          return CON;
54      }
55
56      if (c == '+' || c == '-' || c == '*' || c == '!') {
57          yylval = c;
58          return FID;
59      }
60
61      if (c == EOF) { /* ファイルの終 */
62          exit(0);
63      }
64      /* 上のどの条件にも合わなければ、文字をそのまま返す。 */
65      return c; /* '(', ')', ',', '\n' など */

```

```
66 }
67
68 char* tokenName(int t) { /* デバッグ用 */
69     switch (t) {
70         case '\n': return "(End of Line)";
71         case 256: return "CON";
72         case 257: return "FID";
73         default: return "(Unknown)";
74     }
75 }
76
77 /* token (終端記号) を消費して、次の tokenを読む */
78 void eat(int t) {
79     if (token == t) {
80         token = yylex();
81         return;
82     } else {
83         if (isprint(t)) {
84             printf("eat: Character '%c' is expected at column %d ", t, column);
85         } else {
86             printf("eat: Token %s is expected at column %d ",
87                 tokenName(t), column);
88         }
89         if (isprint(token)) {
90             printf("instead of '%c'.\n", token);
91         } else {
92             printf("instead of %s (code %d).\n", tokenName(token), token);
93         }
94         exit(1);
95     }
96 }
97
98 /* エラーメッセージの出力 */
99 void errorMessage(char* place) {
100     if (isprint(token)) {
101         printf("%s: Unexpected token: '%c' at column %d.\n", place, token, column);
102     } else {
103         printf("%s: Unexpected token: %s (code %d) at column %d.\n",
104             place, tokenName(token), token, column);
105     }
106 }
107 /* 関数プロトタイプ宣言 */
108 void Expr(void);
109 void List(void);
110 void Rest(void);
111
112 /*
113 再帰的構文解析関数群
114 文法の各非終端記号に対応する関数
115 */
116 void Expr(void) {
117     switch (token) {
118         case CON:
119             eat(CON); break;
120         case FID:
121             eat(FID); eat('('); List(); eat(')'); break;
122         default:
123             errorMessage("Expr");
124             exit(1);
125             break;
126     }
127 }
128
129 void List(void) {
130     if (token == CON || token == FID) {
```

```
131     Expr(); Rest();
132 } else {
133     errorMessage("List");
134     exit(1);
135 }
136 }
137
138 void Rest(void) {
139     switch (token) {
140     case ',':
141         eat(','); Expr(); break;
142     case ')':
143         /* do nothing */ break;
144     default:
145         errorMessage("Rest");
146         exit(1);
147         break;
148     }
149 }
150
151 /* 各行の処理 */
152 void processLine(void) {
153     Expr();
154     if (token == '\n') { /* 入力がブロックしないように改行は特別扱い */
155         printf("Correct!\n"); /* eat('\n')の前に出力しておく */
156     }
157     eat('\n');
158 }
159
160 /* main関数 */
161 int main(void) {
162     printf("Ctrl-cで終了します。 \n");
163     token = yylex(); /* 最初のトークンを読む */
164     while (1 /* 無限ループ */) {
165         processLine(); /* 各行を処理する */
166     }
167
168     return 0;
169 }
170
```