```
/*

再帰的下向き構文解析プログラム
（以下の文法に対する構文解析プログラム）
   E    -> C
         | F ( L )
   L    -> E R
   R    -> , E
         | ε

-- 以下は終端記号：字句解析部で処理
   C    -> 0 | 1 | ... | 9   --  一桁の数のみ
   F    -> + | - | * | !


-- 予測型構文解析表
            C       F       (       )       ,       $
   E        C       F ( L ) ×       ×       ×       ×
   L        E R     E R     ×       ×       ×       ×
   R        ×       ×       ×       ε       , E     ×

 */

#include <stdio.h>
#include <stdlib.h>   /* exit() 用 */
#include <ctype.h>    /* isdigit() 用 */

/* 大域変数の宣言 */
int token;      /* 入力の先頭のトークンを表す */
int yylval;     /* token の属性 */

int column = 0; /* デバッグ用 */

/* 終端記号に対応するマクロの定義 */
#define C 256
#define F 257

/* 簡易字句解析ルーチン */
int yylex(void) {
    int c;

    if (token == '\n') { /* 前のトークンが改行だったら */
        column = 0;
    }

    do {
        c = getchar();
        column++;
    } while (c == ' ' || c == '\t');

    if (isdigit(c)) {
        yylval = c - '0'; /* 数字から数へ変換 */
        return C;
    }

    if (c == '+' || c== '-' || c == '*' ||c =='!') {
        yylval = c;
        return F;
    }
```

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

```
119         case C:
120           eat(C); break;
121         case F:
122           eat(F); eat('('); L(); eat(')'); break;
123         default:
124           errMessage("E");
125           exit(1);
126           break;
127       }
128   }
129
130   void L(void) {
131       if (token == C || token == F ) {
132           E(); R();
133       } else {
134           errMessage("L");
135           exit(1);
136       }
137   }
138
139   void R(void) {
140       switch (token) {
141         case ',':
142           eat(','); E(); break;
143         case ')':
144           /* do nothing */ break;
145         default:
146           errMessage("R");
147           exit(1);
148           break;
149       }
150   }
151
152   /* 各行の処理 */
153   void processLine(void) {
154       E();
155       if (token == '\n') { /* 入力がブロックしないように改行は特別扱い */
156           printf("Correct!\n");    /* eat('\n') の前に出力しておく */
157       }
158       eat('\n');
159   }
160
161   /* main関数 */
162   int main(void) {
163       printf("Ctrl-c で終了します。\n");
164       token = yylex(); /* 最初のトークンを読む */
165       while (1 /* 無限ループ */) {
166           processLine();  /* 各行を処理する */
167       }
168
169       return 0;
170   }
171
```