```c
/* ************************************************************ *
 * 演算子順位法による構文解析                                   *
 *                                                              *
 * 1+2*3のような式を構文解析して、計算結果（この場合 7）を出力する *
 *                                                              *
 * 表（prec_tableと op_index)と構文規則（reduce)を書き換えて      *
 * 使用してください。                                           *
 * ************************************************************ */

/* マクロの定義 --- 終端記号・非終端記号を表す定数を定義する */
/* ここからは終端記号、       */
#define BGN 256         /* 始 */
#define END 257         /* 終 */
#define NUM 258         /* 数値 */
#define TERM_MAX 258
/* ここからは非終端記号の定義 */
#define Expr 259

/* 字句解析部が返す``属性''(yylval)の型   *
 *  Yacc (Bison)と同じ形式にする。          */
typedef double YYSTYPE;  /* 使用するトークンの属性の型に応じて変更する。*/
extern YYSTYPE yylval;
/* 字句解析部に flexが生成する関数を用いる場合は、ここまでをヘッダーファ *
 * イルとして分離する。                                                 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

YYSTYPE yylval;

/* ************************************************************ *
 * スタックの実装                                               *
 * ************************************************************ */

struct _elem { /* スタックの要素の型 */
  int token;   /* トークンの種類、259以上は非終端記号 */
  YYSTYPE val; /* 属性値 */
};

typedef struct _elem elem;

elem stack[64]; /* トイプログラムなのでとりあえずスタックの大きさは 64で十分 */

elem* sp = stack; /* 大域変数: スタックポインタ */

void push(int tok, YYSTYPE attr) { /* スタックにプッシュする。 */
  sp->token = tok;
  sp->val   = attr;
  sp++;  /* スタックは下に伸びることに注意 */
}

elem pop(void) { /* スタックをポップする。 */
  if (sp == stack) {
    printf("スタックが空です。¥n");
    return *sp;
  } else {
    sp--;
    return *sp;
  }
}

elem* topmost_token_aux(elem* ptr) { /* topmost_tokenの補助関数 */
  while(ptr->token > TERM_MAX) { /* ptrは非終端記号を指す */
    ptr--;
```

```
66    }
67    /* ptr->token <= TERM_MAX */
68    return ptr;
69  }
70
71  elem* topmost_token(void) { /* スタックの先頭の終端記号 */
72    return topmost_token_aux(sp-1);
73  }
74
75  void debug_token(int t, YYSTYPE v) {
76      switch (t) {
77      case BGN:  printf("BGN"); break;
78      case END:  printf("END"); break;
79      case NUM:  printf("NUM_(%.3f)", v); break;
80      case Expr: printf("Expr_(%.3f)", v); break;
81      default:
82        printf("'%c'", t); break;
83      }
84  }
85
86  void debug_stack(void) { /* デバッグ用: スタックの中身を出力する */
87    elem* sp0;
88
89    printf("スタック 底 <<");
90    for(sp0=stack; sp0<sp; sp0++) {
91          int t =     sp0->token;
92          YYSTYPE v = sp0->val;
93
94          printf(" | ");
95          debug_token(t, v);
96    }
97    printf(" >> 上¥n");
98  }
99
100 /* ************************************************************** *
101  * 字句解析部  (flexが生成する関数に置き換えても良い。)            *
102  * ************************************************************** */
103
104 int yylex(void) { /* 入力の次のトークンを返す。 */
105   int c;
106
107   do {
108     c = getchar();
109   } while (c == ' ' || c == '¥t');  /* 空白を読みとばす */
110
111   if (isdigit(c) || c == '.') {
112     ungetc(c, stdin);
113     scanf("%lf", &yylval);
114     /* ``値''は yylvalという変数に代入して返す。*/
115     return NUM;
116     /* NUMというトークンを返す。*/
117   } else if (c == '¥n') {
118     return END; /* 終りの記号 */
119   } else if (c == EOF) {
120     exit(0);  /* プログラムの終了 */
121   }
122   /* 上のどの条件にも合わなければ、文字をそのまま返す。*/
123   return c;
124 }
125
126 /* ************************************************************** *
127  * 構文解析部                                                     *
128  *                                                                *
129  *   Expr -> NUM                                                  *
130  *         | '(' Expr ')'                                         *
```

```
 *          | Expr '+' Expr                                          *
 *          | Expr '*' Expr                                          *
 * ********************************************************** */

/* ********************************************************** *
 * 演算子順位表の表現     必要に応じて変更する                    *
 * ********************************************************** */
#define LT 0    /* <, Less Than */
#define EQ 1    /* =, Equal */
#define GT 2    /* >, Greater Than */
#define ERR 3   /* エラー, Error */

int op_index(elem* p) { /* 表を引きやすいように連続した数値に写す。*/
  switch (p->token) {
  case BGN:  return 0;
  case '+':  return 1;
  case '*':  return 2;
  case '(':  return 3;
  case ')':  return 4;
  case NUM:  return 5;
  case END:  return 6;
  default:   printf("op_index: 不正な構文要素 (");
             debug_token(p->token, p->val);
             printf(")。¥n");
             exit(1);
             return 0;
  }
}

char prec_table[6][6] = { /* 演算子順位表本体 */
  /* 行に ENDがないこと、列に BGNがないことに注意。*/
          /* '+',  '*',  '(',  ')',  NUM,  END */
  /* BGN */ {LT,   LT,   LT,   ERR,  LT,   EQ  },
  /* '+' */ {GT,   LT,   LT,   GT,   LT,   GT  },
  /* '*' */ {GT,   GT,   LT,   GT,   LT,   GT  },
  /* '(' */ {LT,   LT,   LT,   EQ,   LT,   ERR },
  /* ')' */ {GT,   GT,   ERR,  GT,   ERR,  GT  },
  /* NUM */ {GT,   GT,   ERR,  GT,   ERR,  GT  },
};


/* 演算子順位表を利用する補助関数 */
int prec(elem* left, elem* right) {
  /* leftと rightの関係をprec_tableから引く。 */
  return prec_table[op_index(left)][op_index(right)-1];
}

elem* handle_left(void) { /* 還元が起こる記号の列の左端を見つける */
  elem* next;
  elem* cur = topmost_token(); /* スタックのトップの終端記号の位置 */

  while (1) {
    next = topmost_token_aux(cur-1); /* 次の終端記号の位置 */
    if (prec(next, cur) == LT) {
      return next+1; /* nextの手前が求める場所 */
    } else { /* EQ */
      cur = next;
    }
  }
}

/* ********************************************************** *
 * 構文規則の表現    必要に応じて変更する                      *
 * ********************************************************** */

```

```
196  int reduce(void) {  /* 還元処理 */
197    elem* left = handle_left();   /* 還元する部分の左端を見つける */
198    int num = sp - left;          /* 還元する記号列の長さ */
199    /* printf("reduce:¥t"); */
200
201    switch (num) {                /* どの規則で還元するか? */
202    case 1: {
203      elem data = pop();
204      if (data.token == NUM) {
205        /* Expr -> NUM */
206        printf("reduce: Expr -> NUM_(%.3f)¥n", data.val);
207        push(Expr, data.val);        /* ポップしてすぐプッシュ */
208        break;
209      } else {
210        printf("reduce: 不正なオペランド (");
211        debug_token(data.token, data.val);
212        printf(")。¥n");
213        exit(2);
214      }
215    }
216    case 2: {
217      elem data2 = pop(); elem data1 = pop();
218      printf("reduce: 不正な式 (");
219      debug_token(data1.token, data1.val);
220      printf(",");
221      debug_token(data2.token, data2.val);
222      printf(")。¥n");
223      exit(3);
224    }
225    case 3: {
226      elem data3 = pop(); elem data2 = pop(); elem data1 = pop();
227      if (data1.token == '(' && data2.token == Expr && data3.token == ')') {
228        /* Expr -> '(' Expr ')' */
229        printf("reduce: Expr -> ( Expr )¥n");
230        yylval = data2.val;
231        push(Expr, yylval);
232      } else if (data1.token == Expr && data2.token == '+' && data3.token == Expr) {
233        /* Expr -> Expr + Expr */
234        printf("reduce: Expr -> Expr + Expr¥n");
235        yylval = data1.val+data3.val;
236        push(Expr, yylval);
237      } else if (data1.token == Expr && data2.token == '*' && data3.token == Expr) {
238        /* Expr -> Expr * Expr */
239        printf("reduce: Expr -> Expr * Expr¥n");
240        yylval = data1.val*data3.val;
241        push(Expr, yylval);
242      } else {
243        printf("reduce: 不正な式 (");
244        debug_token(data1.token, data1.val);
245        printf(",");
246        debug_token(data2.token, data2.val);
247        printf(",");
248        debug_token(data3.token, data3.val);
249        printf(")。¥n");
250        exit(4);
251      }
252      break;
253    }
254    default:
255      printf("reduce:構文エラー¥n");
256      exit(5);
257    }
258    debug_stack();
259    return 0;
260  }
```

```
261
262 /* ************************************************************** *
263  * 構文解析関数本体                                               *
264  * ************************************************************** */
265 int yyparse(void) {
266   elem* top;
267   elem next;
268   char relation; /* 関係 */
269
270   push(BGN, 0 /* 0はダミー */); /* 始記号をスタックに積んでおく */
271   next.token = yylex(); /* 入力の最初のトークン */
272   next.val   = yylval;
273   debug_stack();
274   while (1) {
275     top = topmost_token();  /* スタックのトップの終端記号 */
276     if (next.token == END && top->token == BGN) {
277       printf("shift¥n"); /* デバッグ用 */
278           push(next.token, 0 /* ダミー */);
279           debug_stack();
280       printf("終了¥n"); /* デバッグ用 */
281       return 0; /* 成功で終了 */
282     }
283     relation = prec(top, &next);
284     if (relation == LT || relation == EQ) {     /* シフト */
285       printf("shift¥n"); /* デバッグ用 */
286       push(next.token, next.val);
287           debug_stack();
288       next.token = yylex();    /* 次のトークンを読み込む */
289       next.val   = yylval;
290       /* printf ("¥ntoken=%d¥n", next.token); *//* デバッグ用 */
291     } else if (relation == GT) {                /* 還元 */
292       if (reduce()) { /* 0以外は構文エラー */
293         return 1;
294       }
295     } else { /* 表の空欄部分 --- エラー */
296       printf("yyparse: 不正な先読み (");
297       debug_token(next.token, next.val);
298       printf(")。¥n");
299       exit(6);
300     }
301   }
302 }
303
304 int main(void) {
305   while (1) {
306     if(yyparse()==0) { /* 0は正常終了 */
307       printf("答: %g¥n", yylval);
308     }
309   }
310 }
311
```