```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;
/* 字句解析にflexが生成する関数を使う場合は、ここまでをヘッダーとして分離する。*/

#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;
  }
}

```

```c
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--;
  }
  /* ptr->token <= TERM_MAX */
  return ptr;
}

elem* topmost_token(void) { /* スタックの先頭の終端記号 */
  return topmost_token_aux(sp-1);
}

void debug_stack(void) { /* デバッグ用: スタックの中身を出力する */
  elem* sp0;

  printf("スタック 底 <<");
  for(sp0=stack; sp0<sp; sp0++) {
    int t =     sp0->token;
    YYSTYPE v = sp0->val;

    printf(" | ");
    switch (t) {
    case 256: printf("BGN"); break;
    case 257: printf("END"); break;
    case 258: printf("NUM (%.3f)", v); break;
    case 259: printf("Expr (%.3f)", v); break;
    default:
      printf("'%c'", t); break;
    }

  }
  printf(" >> 上¥n");
}

/* ************************************************************** *
 * 字句解析部 (flexが生成する関数に置き換えても良い。)            *
 * ************************************************************** */

int yylex(void) { /* 入力の次のトークンを返す。 */
  int c;

  do {
    c = getchar();
  } while (c == ' ' || c == '¥t');  /* 空白を読みとばす */

  if (isdigit(c) || c == '.') {
    ungetc(c, stdin);
    scanf("%lf", &yylval);
    /* ``値''は yylvalという変数に代入して返す。*/
    return NUM;
    /* NUMというトークンを返す。*/
  } else if (c == '¥n') {
    return END; /* 終りの記号 */
  } else if (c == EOF) {
    exit(0);  /* プログラムの終了 */
  }
  /* 上のどの条件にも合わなければ、文字をそのまま返す。*/
  return c;
}

/* ************************************************************** *
```

```
123   * 構文解析部                                                    *
124   *                                                               *
125   *   Expr -> NUM                                                 *
126   *         | '(' Expr ')'                                        *
127   *         | Expr '+' Expr                                       *
128   *         | Expr '*' Expr                                       *
129   * ************************************************************* */
130
131  /* ************************************************************* *
132   * 演算子順位表の表現     必要に応じて変更する                    *
133   * ************************************************************* */
134  #define LT 0    /* <, Less Than */
135  #define EQ 1    /* =, Equal */
136  #define GT 2    /* >, Greater Than */
137  #define ERR 3   /* エラー, Error */
138
139  int op_index(int token) { /* 表を引きやすいように連続した数値に写す。*/
140    switch (token) {
141    case BGN:  return 0;
142    case '+':  return 1;
143    case '*':  return 2;
144    case '(':  return 3;
145    case ')':  return 4;
146    case NUM:  return 5;
147    case END:  return 6;
148    default:   printf("不正な構文要素 (%d)。¥n", token);
149               exit(1);
150               return 0;
151    }
152  }
153
154  char prec_table[6][6] = { /* 演算子順位表本体 */
155    /* 行に ENDがないこと、列に BGNがないことに注意。*/
156            /* '+',  '*',  '(',  ')',  NUM,  END */
157    /* BGN */ {LT,   LT,   LT,   ERR,  LT,   EQ  },
158    /* '+' */ {GT,   LT,   LT,   GT,   LT,   GT  },
159    /* '*' */ {GT,   GT,   LT,   GT,   LT,   GT  },
160    /* '(' */ {LT,   LT,   LT,   EQ,   LT,   ERR },
161    /* ')' */ {GT,   GT,   ERR,  GT,   ERR,  GT  },
162    /* NUM */ {GT,   GT,   ERR,  GT,   ERR,  GT  },
163  };
164
165
166  /* 演算子順位表を利用する補助関数 */
167  int prec(int left, int right) {
168    /* leftと rightの関係をprec_tableから引く。 */
169    return prec_table[op_index(left)][op_index(right)-1];
170  }
171
172  elem* handle_left(void) { /* 還元が起こる記号の列の左端を見つける */
173    elem* next;
174    elem* cur = topmost_token(); /* スタックのトップの終端記号の位置 */
175
176    while (1) {
177      next = topmost_token_aux(cur-1); /* 次の終端記号の位置 */
178      if (prec(next->token, cur->token) == LT) {
179        return next+1; /* nextの手前が求める場所 */
180      } else { /* EQ */
181        cur = next;
182      }
183    }
```

```
184  }
185
186  /* ************************************************************** *
187   * 構文規則の表現    必要に応じて変更する                          *
188   * ************************************************************** */
189
190  int reduce(void) {  /* 還元処理 */
191    elem* left = handle_left();   /* 還元する部分の左端を見つける */
192    int num = sp - left;          /* 還元する記号列の長さ */
193    /* printf("reduce:¥t"); */
194
195    switch (num) {                /* どの規則で還元するか? */
196    case 1: {
197      elem data = pop();
198      if (data.token == NUM) {
199        /* Expr -> NUM */
200        printf("reduce: Expr -> NUM (%.3f)¥n", data.val);
201        push(Expr, data.val);        /* ポップしてすぐプッシュ */
202        break;
203      } else {
204        printf("エラー: 不正なオペランド (%d)。¥n", data.token);
205        exit(2);
206      }
207    }
208    case 2: {
209      elem data2 = pop(); elem data1 = pop();
210      printf("エラー: 不正な式 (%d, %d)。¥n", data1.token, data2.token);
211      exit(3);
212    }
213    case 3: {
214      elem data3 = pop(); elem data2 = pop(); elem data1 = pop();
215      if (data1.token == '(' && data2.token == Expr && data3.token == ')') {
216        /* Expr -> '(' Expr ')' */
217        printf("reduce: Expr -> ( Expr )¥n");
218        yylval = data2.val;
219        push(Expr, yylval);
220      } else if (data1.token == Expr && data2.token == '+' && data3.token == Expr) {
221        /* Expr -> Expr + Expr */
222        printf("reduce: Expr -> Expr + Expr¥n");
223        yylval = data1.val+data3.val;
224        push(Expr, yylval);
225      } else if (data1.token == Expr && data2.token == '*' && data3.token == Expr) {
226        /* Expr -> Expr * Expr */
227        printf("reduce: Expr -> Expr * Expr¥n");
228        yylval = data1.val*data3.val;
229        push(Expr, yylval);
230      } else {
231        printf("エラー: 不正な式 (%d, %d, %d)。¥n",
232               data1.token, data2.token, data3.token);
233        exit(4);
234      }
235      break;
236    }
237    default:
238      printf("構文エラー¥n");
239      exit(5);
240    }
241    debug_stack();
242    return 0;
243  }
244
```

```
245  /* ********************************************************************** *
246   * 構文解析関数本体                                                       *
247   * ********************************************************************** */
248  int yyparse(void) {
249    int tok;
250    elem* top;
251    char relation; /* 関係 */
252
253    push(BGN, 0 /* 0はダミー */); /* 始記号をスタックに積んでおく */
254    tok = yylex(); /* 入力の最初のトークン */
255    /* printf ("¥ntoken=%d¥n", tok); *//* デバッグ用 */
256    debug_stack();
257    while (1) {
258      top = topmost_token();  /* スタックのトップの終端記号 */
259      if (tok == END && top->token == BGN) {
260        printf("shift¥n"); /* デバッグ用 */
261        push(tok, 0 /* ダミー */);
262        debug_stack();
263        printf("終了¥n"); /* デバッグ用 */
264        return 0; /* 成功で終了 */
265      }
266      relation = prec(top->token, tok);
267      if (relation == LT || relation == EQ) {     /* シフト */
268        printf("shift¥n"); /* デバッグ用 */
269        push(tok, yylval);
270        debug_stack();
271        tok = yylex();    /* 次のトークンを読み込む */
272        /* printf ("¥ntoken=%d¥n", tok); *//* デバッグ用 */
273      } else if (relation == GT) {                /* 還元 */
274        if (reduce()) { /* 0以外は構文エラー */
275          return 1;
276        }
277      } else { /* 表の空欄部分 --- エラー */
278        printf("不正な構文要素 (%d)。¥n", tok);
279        exit(6);
280      }
281    }
282  }
283
284  int main(void) {
285    while (1) {
286      if(yyparse()==0) { /* 0は正常終了 */
287        printf("答: %g¥n", yylval);
288      }
289    }
290  }
291
```