

```

1  /* *****
2  * 演算子順位法による構文解析
3  *
4  * 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
5  *
6  * 表 (prec_table と op_index)と、
7  * 還元規則 (reduce の補助関数の binary_op)を書き換えて
8  * 使用してください。
9  * ***** */
10
11 /* 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 */
12 /* ここからは終端記号、 */
13 #define BGN 256 /* 始 */
14 #define END 257 /* 終 */
15 #define NUM 258 /* 数値 */
16 #define TERM_MAX 258
17 /* ここからは非終端記号の定義 */
18 #define Expr 259
19
20 /* 字句解析部が返す ``属性'' (yyval) の型
21 * Yacc (Bison)と同じ形式にする。 */
22 typedef double YYSTYPE; /* 使用するトークンの属性の型に応じて変更する。*/
23 extern YYSTYPE yylval;
24 /* 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
25 * イルとして分離する。 */
26
27 #include <stdio.h>
28 #include <stdlib.h>
29 #include <ctype.h>
30
31 YYSTYPE yylval;
32
33 /* *****
34 * スタックの実装
35 * ***** */
36
37 struct _elem { /* スタックの要素の型 */
38     int token; /* トークンの種類、259 以上は非終端記号 */
39     YYSTYPE val; /* 属性値 */
40 };
41
42 typedef struct _elem elem;
43
44 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 */
45
46 elem* sp = stack; /* 大域変数: スタックポインタ */
47
48 void push(int tok, YYSTYPE attr) { /* スタックにプッシュする。 */
49     sp->token = tok;
50     sp->val = attr;
51     sp++; /* スタックは下に伸びることに注意 */
52 }
53
54 elem pop(void) { /* スタックをポップする。 */
55     if (sp == stack) {
56         printf("スタックが空です.\n");
57         return *sp;
58     } else {
59         sp--;
60         return *sp;
61     }
62 }
63
64 void clear_stack(void) {
65     sp = stack;
66 }
67
68 elem* topmost_token_aux(elem* ptr) { /* topmost_token の補助関数 */

```

```

69  while (ptr->token > TERM_MAX) { /* ptr は非終端記号を指す */
70      ptr--;
71      if (ptr < stack) {
72          printf("エラー: スタックには終端記号が入っていません.\n");
73          return stack;
74      }
75  }
76  /* ptr->token <= TERM_MAX */
77  return ptr;
78  }
79
80  elem* topmost_token(void) { /* スタックの先頭の終端記号 */
81      return topmost_token_aux(sp-1);
82  }
83
84  void debug_token(int t, YYSTYPE v) {
85      switch (t) {
86          case BGN: printf("BGN"); break;
87          case END: printf("END"); break;
88          case NUM: printf("NUM_(%.3f)", v); break;
89          case Expr: printf("Expr_(%.3f)", v); break;
90          default:
91              printf("%c", t); break;
92      }
93  }
94
95  void debug_stack(void) { /* デバッグ用: スタックの中身を出力する */
96      elem* sp0;
97
98      printf("スタック 底 <<");
99      for (sp0 = stack; sp0 < sp; sp0++) {
100         int t =      sp0->token;
101         YYSTYPE v = sp0->val;
102
103         printf(" | ");
104         debug_token(t, v);
105     }
106     printf(" >> 上\n");
107 }
108
109 /* *****
110 * 字句解析部 (flex が生成する関数に置き換えても良い。)
111 * ***** */
112
113 int yylex(void) { /* 入力の次のトークンを返す。 */
114     int c;
115
116     do {
117         c = getchar();
118     } while (c == ' ' || c == '\t'); /* 空白を読みとばす */
119
120     if (isdigit(c) || c == '.') {
121         ungetc(c, stdin);
122         scanf("%lf", &yylval);
123         /* ``値''は yyval という変数に代入して返す。*/
124         return NUM;
125         /* NUMというトークンを返す。*/
126     } else if (c == '\n') {
127         return END; /* 終りの記号 */
128     } else if (c == EOF) {
129         exit(0); /* プログラムの終了 */
130     }
131     /* 上のどの条件にも合わなければ、文字をそのまま返す。*/
132     return c;
133 }
134
135 /* *****
136 * 構文解析部

```

```

137  *
138  *   Expr -> NUM
139  *         | '(' Expr ')'
140  *         | Expr '+' Expr
141  *         | Expr '*' Expr
142  * ***** */
143
144  /* *****
145  * 演算子順位表の表現    必要に応じて変更する
146  * ***** */
147  #define LT 0    /* <, Less Than */
148  #define EQ 1    /* =, Equal */
149  #define GT 2    /* >, Greater Than */
150  #define ERR 3   /* エラー, Error */
151
152  int op_index(elem* p) { /* 表を引きやすいように連続した数値に写す。*/
153  switch (p->token) {
154  case BGN: return 0;
155  case '+': return 1;
156  case '*': return 2;
157  case '(': return 3;
158  case ')': return 4;
159  case NUM: return 5;
160  case END: return 6;
161  default: printf("op_index: 不正な構文要素 (");
162           debug_token(p->token, p->val);
163           printf(").\n");
164           exit(1);
165           return 0;
166  }
167  }
168
169  char prec_table[6][6] = { /* 演算子順位表本体 */
170  /* 行に END がないこと、列に BGN がないことに注意。*/
171  /* '+', '*', '(', ')', NUM, END */
172  /* BGN */ {LT, LT, LT, ERR, LT, EQ },
173  /* '+' */ {GT, LT, LT, GT, LT, GT },
174  /* '*' */ {GT, GT, LT, GT, LT, GT },
175  /* '(' */ {LT, LT, LT, EQ, LT, ERR },
176  /* ')' */ {GT, GT, ERR, GT, ERR, GT },
177  /* NUM */ {GT, GT, ERR, GT, ERR, GT },
178  };
179
180
181  /* 演算子順位表を利用する補助関数 */
182  int prec(elem* left, elem* right) {
183  /* left と right の関係を prec_table から引く。*/
184  return prec_table[op_index(left)][op_index(right) - 1];
185  }
186
187  elem* handle_left(void) { /* 還元が起こる記号の列の左端を見つける */
188  elem* next;
189  elem* cur = topmost_token(); /* スタックのトップの終端記号の位置 */
190
191  while (1) {
192  next = topmost_token_aux(cur - 1); /* 次の終端記号の位置 */
193  if (prec(next, cur) == LT) {
194  return next + 1; /* next の手前が求める場所 */
195  } else { /* EQ */
196  cur = next;
197  }
198  }
199  }
200
201  /* *****
202  * 構文規則の表現    必要に応じて変更する
203  * ***** */
204

```

```

205 YYSTYPE binary_op(YYSTYPE left, int op, YYSTYPE right) {
206     printf(" 還元: Expr -> Expr "); debug_token(op, 0); printf(" Expr\n");
207     switch (op) {
208         case '+':
209             return left + right;
210         case '*':
211             return left * right;
212         default:
213             printf("処理できない二項演算子: "); debug_token(op, 0); printf("\n");
214             exit(7);
215             return 0;
216     }
217 }
218
219 int reduce(void) { /* 還元処理 */
220     elem* left = handle_left(); /* 還元する部分の左端を見つける */
221     int num = sp - left; /* 還元する記号列の長さ */
222     /* printf("reduce:\t"); */
223
224     switch (num) { /* どの規則で還元するか? */
225     case 1: {
226         elem data = pop();
227         if (data.token == NUM) {
228             /* Expr -> NUM */
229             printf(" 還元: Expr -> NUM_(%.3f)\n", data.val);
230             push(Expr, data.val); /* ポップしてすぐプッシュ */
231             break;
232         } else {
233             printf("reduce: 不正なオペランド (");
234             debug_token(data.token, data.val);
235             printf(")\n");
236             exit(2);
237         }
238     }
239     case 2: {
240         elem data2 = pop(); elem data1 = pop();
241         printf("reduce: 不正な式 (");
242         debug_token(data1.token, data1.val);
243         printf(",");
244         debug_token(data2.token, data2.val);
245         printf(")\n");
246         exit(3);
247     }
248     case 3: {
249         elem data3 = pop(); elem data2 = pop(); elem data1 = pop();
250         if (data1.token == '(' && data2.token == Expr && data3.token == ')') {
251             /* Expr -> '(' Expr ')' */
252             printf(" 還元: Expr -> ( Expr )\n");
253             yylval = data2.val;
254             push(Expr, yylval);
255         } else if (data1.token == Expr && data3.token == Expr) {
256             /* 二項演算子 */
257             yylval = binary_op(data1.val, data2.token, data3.val);
258             push(Expr, yylval);
259         } else {
260             printf("reduce: 不正な式 (");
261             debug_token(data1.token, data1.val);
262             printf(",");
263             debug_token(data2.token, data2.val);
264             printf(",");
265             debug_token(data3.token, data3.val);
266             printf(")\n");
267             exit(4);
268         }
269         break;
270     }
271     default:
272         printf("reduce: 構文エラー\n");

```

```

273     exit(5);
274 }
275 debug_stack();
276 return 0;
277 }
278
279 /* *****
280 * 構文解析関数本体
281 * ***** */
282 int yyparse(void) {
283     elem* top;
284     elem next;
285     char relation; /* 関係 */
286
287     push(BGN, 0 /* 0 はダミー */); /* 始記号をスタックに積んでおく */
288     next.token = yylex(); /* 入力の最初のトークン */
289     next.val = yylval;
290     debug_stack();
291     while (1) {
292         top = topmost_token(); /* スタックのトップの終端記号 */
293
294         printf(" ");
295         debug_token(top->token, top->val);
296         printf(" と ");
297         debug_token(next.token, next.val);
298         printf(" を比較: ");
299
300         if (next.token == END && top->token == BGN) {
301             printf(" シフト\n"); /* デバッグ用 */
302             push(next.token, 0 /* ダミー */);
303             debug_stack();
304             printf(" 終了\n"); /* デバッグ用 */
305             clear_stack();
306             return 0; /* 成功で終了 */
307         }
308
309         relation = prec(top, &next);
310         if (relation == LT || relation == EQ) { /* シフト */
311             printf(" シフト\n"); /* デバッグ用 */
312             push(next.token, next.val);
313             debug_stack();
314             next.token = yylex(); /* 次のトークンを読み込む */
315             next.val = yylval;
316             /* printf ("\ntoken=%d\n", next.token); */ /* デバッグ用 */
317         } else if (relation == GT) { /* 還元 */
318             if (reduce()) { /* 0 以外は構文エラー */
319                 return 1;
320             }
321         } else { /* 表の空欄部分 --- エラー */
322             printf("yyparse: 不正な先読み (");
323             debug_token(next.token, next.val);
324             printf(").\n");
325             exit(6);
326         }
327     }
328 }
329
330 int main(void) {
331     while (1) {
332         if (yyparse() == 0) { /* 0 は正常終了 */
333             printf(" 答: %g\n", yylval);
334         }
335     }
336
337     return 0;
338 }
339

```