数据结构实践课报告

题目：算数表达式求值演示

班级：17052317 姓名：於文卓 学号17061833 完成日期：2018年11月14日

1. **需求分析**

1. 设计一个程序，演示用算符优先法对算术表达式求值的过程

2．以字符序列的形式从终端输入语法正确的、不含变量的整数表达式。利用教科书表3.1给出的算符优先关系，实现对算数四则混合运算表达式的求值，并仿照教科书的例3-1演示在求值中运算符栈、运算数栈、输入字符和主要操作的变化过程。

3. 测试数据：

（1） 3\*(7-2)

（2） 8

（3） 1+2+3+4

（4） 88-1\*5

（5）1024/4\*8

（6）(20+2)\*(6/2)

(7) 3-3-3

（8）8/(9-9)

(9) 2\*(6+2\*(3+6\*(6+6)))

(10) (((6+6)\*6+3)\*2+6)\*2

**概要设计**

1. **栈的抽象数据类型定义为：**

//==========操作数栈=============//

int InitStack\_OPND(OperandStack \*S);

//构造一个空栈S

int GetTop\_OPND(OperandStack \*S, int \*e);

//若栈不为空，则用e返回S的栈顶元素，并返回OK；否则返回FALSE

int Push\_OPND(OperandStack \*S, int e);

//插入元素e为新的栈顶元素

int Pop\_OPND(OperandStack \*S, int \*e);

//若栈S不为空，则删除S的栈顶元素，用e返回其值，并返回OK,否则返回ERROR

//==========操作符栈=============//

int InitStack\_OPTR(OperatorStack \*S);

//构造一个空栈S

char GetTop\_OPTR(OperatorStack \*S);

//若栈不为空，则用e返回S的栈顶元素，并返回OK；否则返回FALSE

int Push\_OPTR(OperatorStack \*S, char e);

//插入元素e为新的栈顶元素

int Pop\_OPTR(OperatorStack \*S, char \*e);

//若栈S不为空，则删除S的栈顶元素，用e返回其值，并返回OK,否则返回ERROR

**2.** **运算操作抽象数据类型定义为：**

int EvalueateExpression(char \*expression);

//算数表达式求值

int Isoperator(char c);

//判断c是否是一个操作符

char Precede(char op1, char op2);

//判断op1和op2优先级的高低，返回'>','<','='

int Operate(int a, char theta, int b);

//对操作数a，b进行theta运算

char \*GetNum( char \*c, int \*num);

//获得以\*c开始的操作数，返回后c为操作符

**本程序包含3个模块：**

(1)主程序模块：

int main(){

初始化；

接受命令；

处理命令；

输出结果；

return 0;

}

(2)操作符栈和操作数模块——实现栈抽象数据类型；

（3）运算单元模块——实现运算操作抽象数据类型；

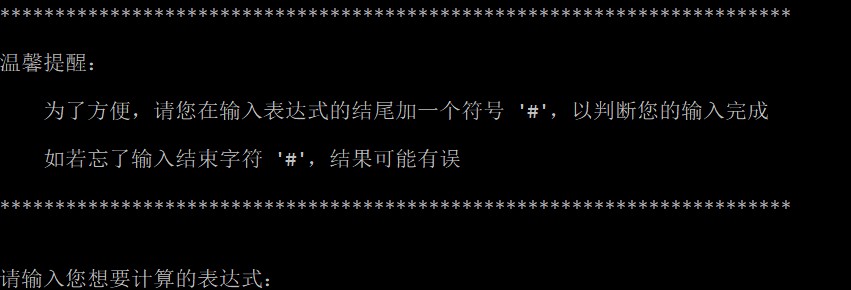
各模块之间的调用关系如下：

主程序模块 栈模块 运算单元模块

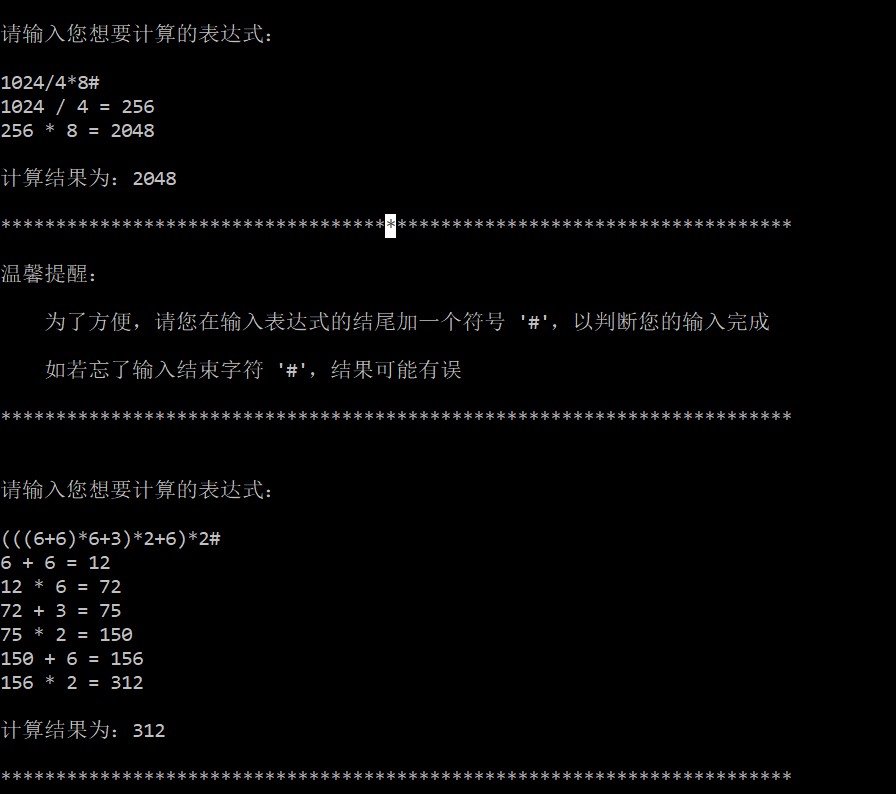
1. **运算单元模块部分实现代码**

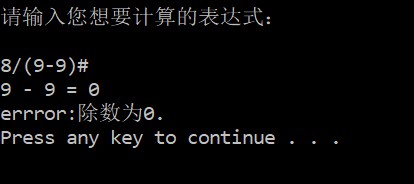
|  |
| --- |
| char \*GetNum(char \*c, int \*num)  {  //获得以\*c开始的操作数，返回后c为操作符  int sum = 0, temp;  while(!Isoperator(\*c)) //当c不是操作符  {  temp = \*c - '0';//字符串转成了数字  /\*  '0' = 0x30;  '1' = 0x31;  char c = '2'  int n = c - '0' = 0x32 - 0x30 = 2;  \*/  sum = sum \* 10 + temp;  c++;  }  \*num = sum;  return c;  }  int Isoperator(char c)  {  //判断c是否是一个运算操作符  switch(c)  {  case '+':  case '-':  case '\*':  case '/':  case '(':  case ')':  case '#':  return TRUE;  break;  default:  return FALSE;  break;  }  }  char Precede(char s\_top, char ch)  {  int i, j;  switch(s\_top)  {  case '+': i = 0; break;  case '-': i = 1; break;  case '\*': i = 2; break;  case '/': i = 3; break;  case '(': i = 4; break;  case ')': i = 5; break;  case '#': i = 6; break;  default: i = 0; break;  }  switch(ch)  {  case '+': j = 0; break;  case '-': j = 1; break;  case '\*': j = 2; break;  case '/': j = 3; break;  case '(': j = 4; break;  case ')': j = 5; break;  case '#': j = 6; break;  default: j = 0; break;  }  return pre[i][j];  }  int Operate(int num\_a, char theta, int num\_b)  {  int result;  switch(theta)  {  case '+':  result = num\_a + num\_b;  break;  case '-':  result = num\_a - num\_b;  break;  case '\*':  result = num\_a \* num\_b;  break;  case '/':  if(num\_b == 0)  {  printf("errror:除数为0.");  exit(0);  }  result = num\_a / num\_b;  break;  default:  printf("输入非法.\n");  break;  }  printf("%d %c %d = %d\n", num\_a, theta, num\_b, result);  return result;  } |

1. **调试分析**
2. **在定义栈时，考虑到操作符是char类型的，而操作数是int类型的，所以需要定义两个栈，操作符栈和操作数栈**
3. **利用算符优先表时，需要把字符转成对应的整数，用switch case可以快速匹配**
4. **按照书上的算法，只能实现0-9的四则运算，需要对连续的数字进行操作**
5. **除法时，考虑除零问题**
6. **关于char转成int，需要减去‘0’**
7. **用户手册**
8. **本程序运行环境为win10操作系统，执行文件为算数表达式求值演示.exe**
9. **用户界面如下**

****

1. **测试数据**

****

****

1. **附件源代码**
2. #include <stdio.h>
3. #include <stdlib.h>
5. #define OK      1
6. #define ERROR   0
7. #define TRUE    1
8. #define FALSE   0
9. #define STACK\_INIT\_SIZE 100
10. #define STACKINCREMENT 10
11. #define EXPRESSIONSIZE 256
12. **char** pre[7][7] =
13. {
14. {'>', '>', '<', '<', '<', '>', '>'},
15. {'>', '>', '<', '<', '<', '>', '>'},
16. {'>', '>', '>', '>', '<', '>', '>'},
17. {'>', '>', '>', '>', '<', '>', '>'},
18. {'<', '<', '<', '<', '<', '=', '0'},
19. {'>', '>', '>', '>', '0', '>', '>'},
20. {'<', '<', '<', '<', '<', '0', '='}
21. };
22. **typedef** **struct** //操作数栈结构定义
23. {
24. **int** \*base;
25. **int** \*top;
26. **int** stacksize;
27. } OperandStack;
29. **typedef** **struct** //操作符栈结构定义
30. {
31. **char** \*base;
32. **char** \*top;
33. **int** stacksize;
34. } OperatorStack;
36. //==========操作数栈=============//
37. **int** InitStack\_OPND(OperandStack \*S);
38. //构造一个空栈S
39. **int** GetTop\_OPND(OperandStack \*S, **int** \*e);
40. //若栈不为空，则用e返回S的栈顶元素，并返回OK；否则返回FALSE
41. **int** Push\_OPND(OperandStack \*S, **int** e);
42. //插入元素e为新的栈顶元素
43. **int** Pop\_OPND(OperandStack \*S, **int** \*e);
44. //若栈S不为空，则删除S的栈顶元素，用e返回其值，并返回OK,否则返回ERROR
46. //==========操作符栈=============//
47. **int** InitStack\_OPTR(OperatorStack \*S);
48. //构造一个空栈S
49. **char** GetTop\_OPTR(OperatorStack \*S);
50. //若栈不为空，则用e返回S的栈顶元素，并返回OK；否则返回FALSE
51. **int** Push\_OPTR(OperatorStack \*S, **char** e);
52. //插入元素e为新的栈顶元素
53. **int** Pop\_OPTR(OperatorStack \*S, **char** \*e);
54. //若栈S不为空，则删除S的栈顶元素，用e返回其值，并返回OK,否则返回ERROR
56. //============运算操作================//
57. **void** Standard(**char** \*expression);
58. //将表达式标准化
59. **int** EvalueateExpression(**char** \*expression);
60. //算数表达式求值
61. **int** Isoperator(**char** c);
62. //判断c是否是一个操作符
63. **char** Precede(**char** op1, **char** op2);
64. //判断op1和op2优先级的高低，返回'>','<','='
65. **int** Operate(**int** a, **char** theta, **int** b);
66. //对操作数a，b进行theta运算
67. **char** \*GetNum( **char** \*c, **int** \*num);
68. //获得以\*c开始的操作数，返回后c为操作符

71. //==========操作数栈==========================================================================//
72. **int** InitStack\_OPND(OperandStack \*S)
73. {
74. //构造一个空操作数栈S
75. S->base = (**int** \*)malloc(STACK\_INIT\_SIZE \* **sizeof**(**int**));
76. **if**(!S->base)//分配失败
77. {
78. printf("分配内存失败.\n");
79. exit(0);
80. }
81. S->top = S->base;//初始化
82. S->stacksize = STACK\_INIT\_SIZE;
83. **return** OK;
84. }
86. **int** GetTop\_OPND(OperandStack \*S, **int** \*e)//获取顶部元素，把得到的数交给e
87. {
88. **if**(S->top == S->base)
89. {
90. printf("栈为空.\n");
91. **return** FALSE;
92. }
93. **else**
94. {
95. \*e = \*(S->top - 1);//s->top所指的地方是空的，所以指针先下移一位
96. **return** OK;
97. }
98. }
100. **int** Push\_OPND(OperandStack \*S, **int** e)
101. {
102. //插入元素e为新的栈顶元素
103. **if**(S->top - S->base >= S->stacksize) //栈已满，追加存储空间
104. {
105. S->base = (**int** \*)realloc(S->base, (S->stacksize + STACKINCREMENT) \* **sizeof**(**int**));
106. **if**(!S->base)
107. {
108. printf("重新申请空间失败.\n");
109. exit(0);
110. }
111. S->top = S->base + S->stacksize; //更改栈顶指针
112. S->stacksize += STACKINCREMENT;
113. }
114. \*S->top = e;
115. S->top++;
116. **return** OK;
117. }
119. **int** Pop\_OPND(OperandStack \*S, **int** \*e)//删掉的数用e存
120. {
121. **if**(S->top == S->base) //栈为空
122. {
123. printf("栈为空.\n");
124. **return** ERROR;
125. }
126. S->top -= 1;
127. \*e = \*(S->top);
128. **return** OK;
129. }
131. //==========操作符栈============================================================================//
132. **int** InitStack\_OPTR(OperatorStack \*S)
133. {
134. //构造一个空操作数栈S
135. S->base = (**char** \*)malloc(STACK\_INIT\_SIZE \* **sizeof**(**char**));
136. **if**(!S->base)//分配失败
137. {
138. printf("分配内存失败.\n");
139. exit(0);
140. }
141. S->top = S->base;
142. S->stacksize = STACK\_INIT\_SIZE;
143. **return** OK;
144. }
146. **char** GetTop\_OPTR(OperatorStack \*S)
147. {
148. //若操作数栈不为空，则返回S的栈顶元素，并返回OK；否则返回FALSE
149. **char** e;
150. **if**(S->top == S->base)
151. {
152. printf("栈为空.\n");
153. }
154. **else**
155. {
156. e = \*(S->top - 1);
157. }
158. **return** e;
159. }
161. **int** Push\_OPTR(OperatorStack \*S, **char** e)
162. {
163. //插入元素e为新的栈顶元素
164. **if**(S->top - S->base >= S->stacksize) //栈已满，追加存储空间
165. {
166. S->base = (**char** \*)realloc(S->base, (S->stacksize + STACKINCREMENT) \* **sizeof**(**char**));
167. **if**(!S->base)
168. {
169. printf("重新申请空间失败.\n");
170. exit(0);
171. }
172. S->top = S->base + S->stacksize; //更改栈顶指针
173. S->stacksize += STACKINCREMENT;
174. }
175. \*S->top++ = e;
176. **return** OK;
177. }
179. **int** Pop\_OPTR(OperatorStack \*S, **char** \*e)
180. {
181. //若栈S不为空，则删除S的栈顶元素，用e返回其值，并返回OK,否则返回ERROR
182. **if**(S->top == S->base) //栈为空
183. {
184. printf("栈为空.\n");
185. **return** ERROR;
186. }
187. \*e = \*(--S->top);
188. **return** OK;
189. }
191. //============运算操作=======================================================================================//
192. **int** EvalueateExpression(**char** \*expression)
193. {
194. **char** \*c = expression;//这里用一个指针去操作字符串
195. OperandStack Operand;
196. OperatorStack Operator;
197. OperandStack \*OPND = &Operand;//operand 操作数
198. OperatorStack \*OPTR = &Operator;//operator 操作符
199. **char** x, theta;
200. **int** a, b, num, result;
201. InitStack\_OPTR(OPTR);//初始化操作符栈
202. InitStack\_OPND(OPND);//初始化操作数栈
203. Push\_OPTR(OPTR, '#'); //首先将匹配符号'#'入栈
204. **while**(\*c != '#' || GetTop\_OPTR(OPTR) != '#')
205. {
206. **if**(\*c == '\0') //遇到回车退出
207. **break**;
208. **if**(!Isoperator(\*c))//如果不是操作符,说明读到了数字
209. {
210. c = GetNum(c, &num);//处理连续的数字，返回的是出去数字的字符串，数字用num存
211. Push\_OPND(OPND, num);
212. }
213. **else**
214. **switch**(Precede(GetTop\_OPTR(OPTR), \*c))//判断优先级
215. {
216. **case** '<'://如果还是新来的那个优先级高，就把新来的加进去
217. Push\_OPTR(OPTR, \*c);
218. c++;//下一个字符串
219. **break**;
220. **case** '=':
221. Pop\_OPTR(OPTR, &x);//如果优先级一样，说明操作符的栈顶是个左括号，不仅不加入这个右括号，还把左括号退栈
222. c++;
223. **break**;
224. **case** '>'://如果还是原来那个操作符的优先级大，把原来那个操作做掉 具体做法是 操作符栈退一个栈 操作数栈退两个栈
225. Pop\_OPTR(OPTR, &theta);
226. Pop\_OPND(OPND, &b);
227. Pop\_OPND(OPND, &a);
228. result = Operate(a, theta, b);
229. Push\_OPND(OPND, result);
230. **break**;
231. **default**: **break**;
232. }
233. }
234. GetTop\_OPND(OPND, &result);//栈里面最后一个值就是结果
235. **return** result;
236. }
238. **char** \*GetNum(**char** \*c, **int** \*num)
239. {
240. //获得以\*c开始的操作数，返回后c为操作符
241. **int** sum = 0, temp;
242. **while**(!Isoperator(\*c)) //当c不是操作符
243. {
244. temp = \*c - '0';//字符串转成了数字
245. /\*
246. '0' = 0x30;
247. '1' = 0x31;
248. char c = '2'
249. int n = c - '0' = 0x32 - 0x30 = 2;
250. \*/
251. sum = sum \* 10 + temp;
252. c++;
253. }
254. \*num = sum;
255. **return** c;
256. }
258. **int** Isoperator(**char** c)
259. {
260. //判断c是否是一个运算操作符
261. **switch**(c)
262. {
263. **case** '+':
264. **case** '-':
265. **case** '\*':
266. **case** '/':
267. **case** '(':
268. **case** ')':
269. **case** '#':
270. **return** TRUE;
271. **break**;
272. **default**:
273. **return** FALSE;
274. **break**;
275. }
276. }

279. //-------------------------------------------方法一-----------------------------------------------
280. **char** Precede(**char** s\_top, **char** ch)
281. {
282. **int** i, j;
283. **switch**(s\_top)
284. {
285. **case** '+': i = 0; **break**;
286. **case** '-': i = 1; **break**;
287. **case** '\*': i = 2; **break**;
288. **case** '/': i = 3; **break**;
289. **case** '(': i = 4; **break**;
290. **case** ')': i = 5; **break**;
291. **case** '#': i = 6; **break**;
292. **default**: i = 0; **break**;
293. }
294. **switch**(ch)
295. {
296. **case** '+': j = 0; **break**;
297. **case** '-': j = 1; **break**;
298. **case** '\*': j = 2; **break**;
299. **case** '/': j = 3; **break**;
300. **case** '(': j = 4; **break**;
301. **case** ')': j = 5; **break**;
302. **case** '#': j = 6; **break**;
303. **default**: j = 0; **break**;
304. }
305. **return** pre[i][j];
306. }

309. //-------------------------------------------方法二-------------------------------------------------
310. // char Precede(char op1, char op2) //op1是已经在栈里面的了，op2是后来加进来的  这里参考了网上的方法
311. // {
312. //     //判断op1和op2优先级的高低，返回'>','<','='
313. //     switch(op1)
314. //     {
315. //     case '+':
316. //         switch(op2)
317. //         {
318. //         case '\*':
319. //         case '/':
320. //         case '(':
321. //             return '<';
322. //             break;
323. //         default:
324. //             return '>';
325. //             break;
326. //         }
327. //         break;
328. //     case '-':
329. //         switch(op2)
330. //         {
331. //         case '\*':
332. //         case '/':
333. //         case '(':
334. //             return '<';
335. //             break;
336. //         default:
337. //             return '>';
338. //             break;
339. //         }
340. //         break;
341. //     case '\*':
342. //         switch(op2)
343. //         {
344. //         case '(':
345. //             return '<';
346. //             break;
347. //         default:
348. //             return '>';
349. //             break;
350. //         }
351. //         break;
352. //     case '/':
353. //         switch(op2)
354. //         {
355. //         case '(':
356. //             return '<';
357. //             break;
358. //         default:
359. //             return '>';
360. //             break;
361. //         }
362. //         break;
363. //     case '(':
364. //         switch(op2)
365. //         {
366. //         case ')':
367. //             return '=';
368. //             break;
369. //         default:
370. //             return '<';
371. //             break;
372. //         }
373. //         break;
374. //     case ')':
375. //         switch(op2)
376. //         {
377. //         default:
378. //             return '>';
379. //             break;
380. //         }
381. //         break;
382. //     case '#':
383. //         switch(op2)
384. //         {
385. //         case '#':
386. //             return '=';
387. //             break;
388. //         default:
389. //             return '<';
390. //             break;
391. //         }
392. //         break;
393. //     default:
394. //         return '<';
395. //         break;
396. //     }
397. // }
399. **int** Operate(**int** num\_a, **char** theta, **int** num\_b)
400. {
401. **int** result;
402. **switch**(theta)
403. {
404. **case** '+':
405. result = num\_a + num\_b;
406. **break**;
407. **case** '-':
408. result = num\_a - num\_b;
409. **break**;
410. **case** '\*':
411. result = num\_a \* num\_b;
412. **break**;
413. **case** '/':
414. **if**(num\_b == 0)
415. {
416. printf("errror:除数为0.");
417. exit(0);
418. }
419. result = num\_a / num\_b;
420. **break**;
421. **default**:
422. printf("输入非法.\n");
423. **break**;
424. }
425. printf("%d %c %d = %d\n", num\_a, theta, num\_b, result);
426. **return** result;
428. }
430. //------------------------------------------main---------------------
432. **int** main()
433. {
434. **while**(1)
435. {
437. **int** result = 0;
438. **char** \*expression = (**char** \*)malloc(**sizeof**(**char**) \* EXPRESSIONSIZE);
439. **if**(expression == NULL)
440. {
441. printf("内存分配失败\n");
442. exit(0);
443. }
445. printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");
446. puts("温馨提醒：\n");
447. puts("    为了方便，请您在输入表达式的结尾加一个符号 '#'，以判断您的输入完成");
448. puts("\n    如若忘了输入结束字符 '#'，结果可能有误");
449. printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");
450. puts("\n请输入您想要计算的表达式：\n");
451. gets(expression);
452. result = EvalueateExpression(expression);
453. printf("\n计算结果为：%d\n", result);
454. }
455. **return** 0;

458. }