

CS205 Assignment 2 Report

SID: 11910718

Name: 陆彦青

Part 1. Introduction & Features

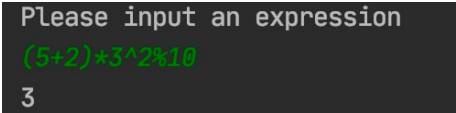
1. This is a simple calculator that supports the operations of addition, subtraction, multiplication, division, exponentiation, square root, mod and factorial for positive and negative integers and decimals.
2. It also supports the definition of variables(all the letters can be the name of variables) and the assignment of variable's value by the values of other variables.
3. Specifically, it can support arbitrary precision for addition, subtraction and multiplication.
4. Some math functions such as **sqrt()** are supported.
5. The postfix expression and recursion are used to complete the functions.

Part 2. Result & Verification

Test case #1:

```
1   Input:  (5+2)*3^2%10
2   Output: 3
```

Screen-shot for case #1:

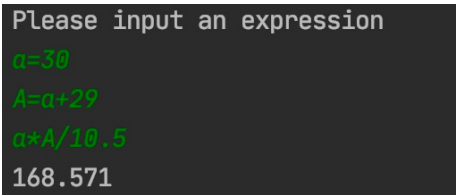


```
Please input an expression
(5+2)*3^2%10
3
```

Test case #2:

```
1   Input:  a=30
2           A=a+29
3           a*A/10.5
4   Output: 168.571
```

Screen-shot for case #2:



```
Please input an expression
a=30
A=a+29
a*A/10.5
168.571
```

Test case #3:

```
1   Input:  a=5
2           a!-5!-255
3   Output: -255
```

Screen-shot for case #3:

```
Please input an expression
a=5
a!-5!-255
-255
```

Test case #4:

```
1   Input:  3888888888888888888*0.00000000005+11111111111111111111
2   Output: 11111111111111111111305555555.4444444444
```

Screen-shot for case #4:

```
Please input an expression
3888888888888888888*0.00000000005+11111111111111111111
11111111111111111111305555555.4444444444
```

Test case #5:

```
1   Input:  s=1
2           q=2
3           r=3
4           t=2
5           sqrt((s+q*r)*t-r-2)
6   Output: 3
```

Screen-shot for case #5:

```
Please input an expression
s=1
q=2
r=3
t=2
sqrt((s+q*r)*t-r-2)
3
Please input an expression
q
Process finished with exit code 0
```

Part 3. Difficulties & Solutions

1. Switch the infix expression into postfix expression. My solution is to classify the different cases of input and use the **stack** to handle them separately.
2. Support the definition of variables and assign the value of a variable with the value of existing variables. I create an array to store the value of different letters and use **recursion** in the function **toPostfix**.
3. Compute the output value of the expression. According to the laws of calculating a postfix expression, I use the **stack** again.
4. Support the operation of arbitrary precision. Since the range of **int, long** is limited, I create the new functions **add**, **sub** and **mul** to switch the numbers to **string** and compute them according to the fundamental laws of operation.
5. Recognize math functions. The function **replaceAll** is used to replace all the math functions in an expression to unused characters.
6. Support the operations of multibit numbers. My solution is to insert the separators(',') between complete numbers and operators. Then I create a function **split(const string &src)** to split the return value of **toPostfix** into a **vector<string>**. Finally, I can use the function **compute(string &exp)** to get the output value.

Part 4. Appendix(Source Code)

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <stack>
5  #include <cmath>
6
7  using namespace std;
8
9  vector<string> split(const string &src)//将后缀表达式分割为vector
10 {
11     vector<string> dest;
12     string::size_type start = 0, index;
13     string substring;
14     index = src.find_first_of(',', start);
15     while (index != string::npos && start < src.size() - 1) {
16         substring = src.substr(start, index - start);
17         dest.push_back(substring);
18         start = index + 1;
19         index = src.find_first_of(',', start);
20     }
21     if (start < src.size()) {
22         substring = src.substr(start);
23         dest.push_back(substring);
24     }
25     return dest;
26 }
27
28
29 string& replaceAll(string& str, const string& old, const string& new_value){//替换全部指定字符串,
    便于计算某些math function
30     while (str.find(old) != string::npos){
31         str.replace(str.find(old), old.size(), new_value);
32     }
33     return str;
34 }
35
36 double alphabet[52] = {0};
37
38 void defineVar(char letter, double num) {
39     if (letter - 'a' < 0) {
40         alphabet[letter - 'A'] = num;
41     } else {
42         alphabet[letter - 'a' + 26] = num;
43     }
44 }
45
46
47 bool isNumber(const string& str) { //判断表达式中的数字
48     for (int i = 0; i < str.size(); i++) {
```

```

49     if (str[i] == '.' && i < str.size() - 1 && str[i + 1] >= '0' && str[i + 1] <= '9') {
50         continue;
51     }
52     if (str[i] == '-' && str.size() > 1) {
53         continue;
54     }
55     if (str[i] > '9' || str[i] < '0')
56         return false;
57 }
58 return true;
59 }
60
61 bool isInt(const string& str){ //判断是否为整数
62     if (str.find('.') == string::npos) return true;
63     else return false;
64 }
65
66 string add(string& a, string& b){ //高精度加法
67     string sub(string& a, string b);
68     string str;
69     if (a[0] == '-'){
70         if (b[0] == '-'){
71             str = '-' + add(a.erase(0,1),b.erase(0,1));
72             return str;
73         } else {
74             return sub(b,a.erase(0,1));
75         }
76     } else {
77         if (b[0] == '-'){
78             return sub(a,b.erase(0,1));
79         } else {
80             string :: size_type a_int = a.find('.'),b_int = b.find('.'); //补0使位数相等
81             string :: size_type a_dec,b_dec;
82             if (a_int == string::npos){
83                 a_int = a.size();
84                 a_dec = 0;
85             } else {
86                 a_dec = a.size()-a_int-1;
87             }
88             if (b_int == string::npos){
89                 b_int = b.size();
90                 b_dec = 0;
91             } else {
92                 b_dec = b.size()-b_int-1;
93             }
94             if (a_int > b_int){
95                 for (int i = 0; i < a_int-b_int; ++i)
96                     b = '0' + b;
97             } else if (a_int < b_int){
98                 for (int i = 0; i < b_int-a_int; ++i)
99                     a = '0' + a;
100             }

```

```

101         if (a_dec > b_dec){
102             if (b_dec == 0) b += '.';
103             for (int i = 0; i < a_dec-b_dec; ++i)
104                 b += '0';
105         } else if (a_dec < b_dec){
106             if (a_dec == 0) a += '.';
107             for (int i = 0; i < b_dec-a_dec; ++i)
108                 a += '0';
109         }
110
111         int carry = 0, res;
112         for (int i = a.size()-1; i > -1; --i) {
113             if (a[i] == '.'){
114                 str = '.' + str;
115             } else {
116                 char c = a[i], d = b[i];
117                 res = (c - '0' + d - '0' + carry) % 10;
118                 carry = (c - '0' + d - '0' + carry) / 10;
119                 str = char(res + '0') + str;
120             }
121         }
122         if (carry != 0) str = char(carry + '0') + str;
123
124         return str;
125     }
126 }
127 }
128 string sub(string& a, string b){ //高精度减法
129     string str;
130     if (b[0] == '-'){
131         return add(a, b.erase(0, 1));
132     } else {
133         if (a[0] == '-'){
134             str = '-' + add(a.erase(0, 1), b);
135             return str;
136         } else {
137             if (stod(a) < stod(b)) return '-' + sub(b, a);
138             else {
139                 string::size_type a_int = a.find('.'), b_int = b.find('.'); //补0使位数相等
140                 string::size_type a_dec, b_dec;
141                 if (a_int == string::npos) {
142                     a_int = a.size();
143                     a_dec = 0;
144                 } else {
145                     a_dec = a.size() - a_int - 1;
146                 }
147                 if (b_int == string::npos) {
148                     b_int = b.size();
149                     b_dec = 0;
150                 } else {
151                     b_dec = b.size() - b_int - 1;
152                 }

```

```

153         if (a_int > b_int) {
154             for (int i = 0; i < a_int - b_int; ++i)
155                 b = '0' + b;
156         } else if (a_int < b_int) {
157             for (int i = 0; i < b_int - a_int; ++i)
158                 a = '0' + a;
159         }
160         if (a_dec > b_dec) {
161             if (b_dec == 0) b += '.';
162             for (int i = 0; i < a_dec - b_dec; ++i)
163                 b += '0';
164         } else if (a_dec < b_dec) {
165             if (a_dec == 0) a += '.';
166             for (int i = 0; i < b_dec - a_dec; ++i)
167                 a += '0';
168         }
169
170         int carry = 0, res;
171         for (int i = a.size() - 1; i > 0; --i) {
172             char c = a[i], d = b[i];
173             if (a[i] == '.') {
174                 str = '.' + str;
175             } else {
176                 res = c - d - carry;
177                 if (res < 0) {
178                     res += 10;
179                     carry = 1;
180                 } else carry = 0;
181                 str = char(res + '0') + str;
182             }
183         }
184         if (a[0] != '.') {
185             res = a[0] - b[0] - carry;
186             str = to_string(res) + str;
187         } else if (carry == 0) {
188             str = '.' + str;
189         } else {
190             str = "-1." + str;
191         }
192         return str;
193     }
194 }
195 }
196 }
197
198 string mul(string& a, string& b){ //高精度乘法
199     if (a[0] == '-'){
200         if (b[0] == '-'){
201             return mul(a.erase(0,1),b.erase(0,1));
202         } else {
203             return '-' + mul(a.erase(0,1),b);
204         }

```

```

205 } else {
206     if (b[0] == '-') {
207         return '-' + mul(a, b.erase(0, 1));
208     } else {
209         string::size_type a_int = a.find('.'), b_int = b.find('.');
210         string::size_type a_dec = 0, b_dec = 0;
211         if (a_int != string::npos) {
212             a = a.erase(a_int, 1);
213             a_dec = a.size() - a_int;
214         }
215         if (b_int != string::npos) {
216             b = b.erase(b_int, 1);
217             b_dec = b.size() - b_int;
218         }
219         string rows[b.size()];
220         for (int i = b.size() - 1; i >= 0; --i) {
221             string row;
222             int over = 0;
223             for (int j = a.size() - 1; j >= 0; --j) {
224                 string tmp = to_string((a[j] - '0') * (b[i] - '0') + over);
225                 if (tmp.size() > 1) {
226                     row = tmp[1] + row;
227                     over = tmp[0] - '0';
228                 } else {
229                     row = tmp + row;
230                     over = 0;
231                 }
232             }
233             if (over != 0) row = char(over + '0') + row;
234             int k = i;
235             while ((b.size() - k - 1) > 0) {
236                 row += '0';
237                 k++;
238             }
239             rows[i] = row;
240         }
241         string result = "0";
242         for (int i = 0; i < b.size(); ++i) {
243             result = add(result, rows[i]);
244         }
245         string::size_type num = a_dec + b_dec;
246         if (num != 0) {
247             result = result.insert(result.size() - num, ".");
248             while (result[result.size() - 1] == '0' || result[result.size() - 1] == '.') result.
                pop_back(); //去掉多余的0
249         }
250         while (result[0] == '0') result.erase(0, 1);
251         if (result[0] == '.') result = '0' + result;
252         return result;
253     }
254 }
255 }

```



```

256 string div(string& a, string& b){
257     return to_string(stod(a) / stod(b));
258 }
259 string mod(string& a, string& b){
260     return to_string((int)stod(a) % (int)stod(b));
261 }
262
263 string factorial(const string& in) {
264     string in1 = in;
265     string out = "1";
266     if (in[0] != '-' && isInt(in)) { //负数、小数无法进行阶乘运算
267         while (sub(in1, "1") != "0") {
268             out = mul(out, in1);
269             in1 = sub(in1, "1");
270         }
271         return out;
272     }
273     return in;
274 }
275
276 bool flag = true;
277
278 string compute(string &exp) //根据后缀表达式的规则计算
279 {
280     vector<string> transform = split(exp);
281     stack<string> tmp;
282     string a, b;
283     if (transform.size() == 1) {
284         return transform[0];
285     }
286     for (string str : transform) {
287         if (isNumber(str)) {
288             tmp.push(str);
289         } else if (str == "!") {
290             a = tmp.top();
291             tmp.pop();
292             tmp.push(factorial(a));
293         } else if (str == "@") {
294             double x = stod(tmp.top());
295             tmp.pop();
296             tmp.push(to_string(sqrt(x)));
297         } else {
298             char opr = str[0];
299             // cout << opr << endl;
300             b = tmp.top();
301             tmp.pop();
302             a = tmp.top();
303             tmp.pop();
304             switch (opr) {
305                 case '+':
306                     tmp.push(add(a, b));
307                     break;

```

```

308         case '-':
309             tmp.push(sub(a,b));
310             break;
311         case '*':
312             tmp.push(mul(a,b));
313             break;
314         case '/':
315             tmp.push(div(a,b));
316             break;
317         case '%':
318             if (!isInt(b)) {
319                 cout << "This kind of operation is invalid" << endl;
320                 flag = false;
321                 break;
322             }
323             tmp.push(mod(a,b));
324             break;
325         case '^':
326             if (!isInt(a) || isInt(b)) {
327                 cout << "This kind of operation is invalid" << endl;
328                 flag = false;
329                 break;
330             }
331             double c,d;
332             c = stod(a), d = stod(b);
333             tmp.push(to_string(pow(c, d)));
334             break;
335         default :
336             cout << "Your input contains invalid character" << endl;
337             flag = false;
338     }
339 }
340 }
341
342 return tmp.top();
343
344 }
345
346 string toPostfix(string &in) { //将中缀表达式转化为后缀表达式(只在赋值时使用此方法)
347     stack<char> s1;
348     string out;
349     in = replaceAll(in,"sqrt", "@");
350
351     for (int i = 0; i < in.size(); i++) {
352         if ((in[i] > 64 && in[i] < 91) || (in[i] > 96 && in[i] < 123)) {
353             string letter;
354             if (in[i] - 'a' < 0) {
355                 letter = to_string(alphabet[in[i] - 'A']);
356             } else {
357                 letter = to_string(alphabet[in[i] - 'a' + 26]);
358             }
359             if (letter[0] == '-') {

```

```

360         out += "0,";
361         out += letter.substr(1);
362         out += ',';
363         s1.push('-');
364     } else if (letter[0] == '+') {
365         out += "0,";
366         out += letter.substr(1);
367         out += ',';
368         s1.push('+');
369     } else {
370         out += letter;
371         out += ',';
372     }
373     //cout << out << endl;
374 } else
375     switch (in[i]) {
376         case ')':
377             while (!s1.empty() && s1.top() != '(') {
378                 out += s1.top();
379                 out += ',';
380                 s1.pop();
381             }
382             s1.pop();
383             if (s1.top() == '@'){
384                 out += "@,";
385                 s1.pop();
386             }
387             break;
388
389         case '(':
390             s1.push(in[i]);
391             break;
392         case '!':
393             out += "!,";
394             break;
395         case '*':
396         case '/':
397         case '%':
398             while (!s1.empty() && s1.top() != '+' && s1.top() != '-' && s1.top() != '(') {
399                 out += s1.top();
400                 out += ',';
401                 s1.pop();
402             }
403             s1.push(in[i]);
404             break;
405         case '-':
406         case '+':
407             if (i == 0 || in[i - 1] == '(' || in[i - 1] == '*' || in[i - 1] == '/' ||
408                 in[i - 1] == '+' || in[i - 1] == '-') //判断+-代表正负or运算符
409             {
410                 out += in[i];
411                 break;

```

```

412         }
413         while (!s1.empty() && s1.top() != '(') {
414             out += s1.top();
415             out += ',';
416             s1.pop();
417         }
418         s1.push(in[i]);
419         break;
420     case '0':
421     case '1':
422     case '2':
423     case '3':
424     case '4':
425     case '5':
426     case '6':
427     case '7':
428     case '8':
429     case '9':
430         out += in[i];
431         if (i == in.size() - 1 || in[i + 1] != '.' && (in[i + 1] < '0' || in[i + 1] > '
9')) {
432             out += ',';
433         }
434         break;
435     case '.':
436         out += in[i];
437         break;
438     default:
439         s1.push(in[i]);
440         break;
441     }
442
443 }
444 while (!s1.empty()) {
445     out += s1.top();
446     out += ',';
447     s1.pop();
448 }
449 // cout << out << endl;
450 return out;
451 }
452
453 string toPostfix() { //重载上一个方法,function中输入便于实现递归
454     cin.clear();
455     string in;
456     cin >> in;
457     if (in == "q") //输入q表示退出程序
458     {
459         return "end";
460     }
461     stack<char> s1;
462     string out;

```

```

463 in = replaceAll(in, "sqrt", "@");
464
465 for (int i = 0; i < in.size(); i++) {
466     if ((in[i] > 64 && in[i] < 91) || (in[i] > 96 && in[i] < 123)) {
467         string letter;
468         //cout << "letterMode" << endl;
469         if (in[i] - 'a' < 0) {
470             letter = to_string(alphabet[in[i] - 'A']);
471         } else {
472             letter = to_string(alphabet[in[i] - 'a' + 26]);
473         }
474         if (letter[0] == '-') {
475             out += "0,";
476             out += letter.substr(1);
477             out += ','; // ',' 是分隔符
478             s1.push('-');
479         } else if (letter[0] == '+') {
480             out += "0,";
481             out += letter.substr(1);
482             out += ',';
483             s1.push('+');
484         } else {
485             out += letter;
486             out += ',';
487         }
488     } else
489         switch (in[i]) {
490             case '=': //实现定义变量及连环定义，利用了递归的思想
491                 if (i > 0 || i < in.size() - 1) {
492                     string subStr = in.substr(i + 1);
493                     string subRes = toPostfix(subStr);
494                     string tmpVal = compute(subRes);
495                     defineVar(in[i - 1], stod(tmpVal));
496                     return toPostfix();
497                 }
498                 break;
499             case ')':
500                 while (!s1.empty() && s1.top() != '(') {
501                     out += s1.top();
502                     out += ',';
503                     s1.pop();
504                 }
505                 s1.pop();
506                 if (s1.top() == '@'){
507                     out += "0,";
508                     s1.pop();
509                 }
510                 break;
511             case '(':
512                 s1.push(in[i]);
513                 break;

```

```

515         case '!':
516             out += "!,";
517             break;
518         case '*':
519         case '/':
520         case '%':
521             while (!s1.empty() && s1.top() != '+' && s1.top() != '-' && s1.top() != '(') {
522                 out += s1.top();
523                 out += ',';
524                 s1.pop();
525             }
526             s1.push(in[i]);
527             break;
528         case '-':
529         case '+':
530             if (i == 0 || in[i - 1] == '(' || in[i - 1] == '*' || in[i - 1] == '/' ||
531                 in[i - 1] == '+' || in[i - 1] == '-') //判断+-代表正负or运算符
532             {
533                 out += in[i];
534                 break;
535             }
536             while (!s1.empty() && s1.top() != '(') {
537                 out += s1.top();
538                 out += ',';
539                 s1.pop();
540             }
541             s1.push(in[i]);
542             break;
543         case '0':
544         case '1':
545         case '2':
546         case '3':
547         case '4':
548         case '5':
549         case '6':
550         case '7':
551         case '8':
552         case '9':
553             out += in[i];
554             if (i == in.size() - 1 || in[i + 1] != '.' && (in[i + 1] < '0' || in[i + 1] > '
555                 9')) {
556                 out += ','; //加入分隔符, 便于实现多位数运算
557             }
558             break;
559         case '.':
560             out += in[i];
561             break;
562         default:
563             s1.push(in[i]);
564             break;
565     }

```

```

566     }
567     while (!s1.empty()) {
568         out += s1.top();
569         out += ',';
570         s1.pop();
571     }
572     // cout << out << endl;
573     return out;
574 }
575
576 int main() {
577     while (true) {
578         cout << "Please input an expression" << endl;
579         string res = toPostfix();
580         if (res == "end") {
581             break;
582         }
583         string result = compute(res);
584         if (flag) {
585             cout << result << endl;
586         }
587         flag = true;
588     }
589
590     return 0;
591 }

```