

```

1: #include <iostream>
2: #include <cstring>
3: #include <cstdlib>
4: #include <cmath>
5:
6: using namespace std;
7:
8:
9: // data structure -----
10:
11: struct Data {
12:     float value;
13:     char op;
14:     bool is_op;
15: };
16:
17: class stack
18: {
19:     public:
20:         stack () ;
21:         void push (Data) ;
22:         Data pop () ;
23:         bool empty () ;
24:
25:     private:
26:         Data array[1000];
27:         int index;
28:
29: };
30:
31: // ***** Write Your Code Here ***** //
32: stack::stack(){
33:     index=0;
34: };
35: void stack::push(Data data){
36:     array[index]=data;
37:     index++;
38:
39: };
40: Data stack::pop(){
41:     index--;
42:     return array[index];
43:
44: };
45: bool stack::empty(){
46:     bool emp;
47:     emp=(index==0)?1:0;
48:     return emp;
49: };
50: // pass
51: // *****
52:
53: // function declarations -----
54:

```

```

55: int convert (char [], Data []) ;
56:
57: // functions for algorithm -----
58:
59: float calculate (float a, char op, float b) {
60:     if (op == '+')
61:         return a + b;
62:     else if (op == '-')
63:         return a - b;
64:     else if (op == '*')
65:         return a * b;
66:     else
67:         return a / b;
68: }
69:
70:
71: void evaluate (stack &stk) {
72:
73:     // Fetch the partial expression from stk
74:     stack stk_tmp;
75:     Data data;
76:     // ***** Write Your Code Here ***** //
77:     // pass
78:     while(!stk.empty())
79:     {
80:         data=stk.pop();
81:         if(data.is_op&&data.op=='('){
82:             break;
83:         }
84:         stk_tmp.push(data);
85:     }
86:     // ***** //
87:
88:     // Evaluate the partial expression
89:     float sum1 = 0;
90:     char op_cur = '+';
91:     float sum2 = stk_tmp.pop().value;
92:     while ( !stk_tmp.empty() ) {
93:         char op_nxt = stk_tmp.pop().op;
94:         if (op_nxt == '+' || op_nxt == '-') {
95:             sum1 = calculate(sum1, op_cur, sum2);
96:             op_cur = op_nxt;
97:             sum2 = stk_tmp.pop().value;
98:         }
99:         else
100:             sum2 = calculate(sum2, op_nxt, stk_tmp.pop().value);
101:     }
102:     data.value = calculate(sum1, op_cur, sum2);
103:     data.is_op = false;
104:     stk.push(data);
105:
106: }
107:
108: // main function -----

```

```

109:
110: int main ()
111: {
112:     while (true) {
113:         int a=0;
114:         // Prompt user to input the expression
115:         char expr_c[1000];
116:         cout << "Enter the expression: ";
117:         cin.getline(expr_c, 1000);
118:
119:         // Exit if the expression is "exit"
120:         if ( !strcmp(expr_c, "exit") )
121:             break;
122:
123:         // Covert character array to our format
124:         Data expr[1000];
125:         int expr_size = convert(expr_c, expr);
126:
127:         // Evaluate the expression
128:         stack stk;
129:         // ***** Write Your Code Here ***** //
130:         // pass
131:         for(int i=0;i<expr_size;i++)
132:         {
133:             if(expr[i].is_op&&expr[i].op=='')
134:             {
135:                 evaluate(stk);
136:             }
137:             else{
138:                 stk.push(expr[i]);
139:             }
140:         }
141:         evaluate(stk);
142:         // ***** //
143:
144:         // Show out the result
145:         cout << "Result: " << stk.pop().value << endl;
146:
147:     }
148:     return 0;
149: }
150:
151: // function for converting expression format -----
152:
153: int convert (char expr_c[], Data expr[]) {
154:     int j = 0;
155:     bool sign_flag = false;
156:     if (expr_c[0] == '+' || expr_c[0] == '-')
157:         sign_flag = true;
158:     for (int i = 0; i < strlen(expr_c); ++i) {
159:         if (expr_c[i] == ' ')
160:             continue;
161:         expr[j].is_op = true;
162:         static bool negative_flag = false;

```

```

163:         if ( expr_c[i] == '(' || expr_c[i] == ')' || expr_c[i] == '+' ||
164:             expr_c[i] == '-' || expr_c[i] == '*' || expr_c[i] == '/' ) {
165:             if ( expr_c[i] == '+' || expr_c[i] == '-' ||
166:                 expr_c[i] == '*' || expr_c[i] == '/' ) {
167:                 if (sign_flag) {
168:                     if (expr_c[i] == '-')
169:                         negative_flag = true;
170:                     continue;
171:                 }
172:                 sign_flag = true;
173:             }
174:             expr[j++].op = expr_c[i];
175:         }
176:     else {
177:         expr[j].is_op = false;
178:         char n_c[10];
179:         int k = i;
180:         while ( expr_c[k] != '(' && expr_c[k] != ')' && expr_c[k] != '+' &&
181:             expr_c[k] != '-' && expr_c[k] != '*' && expr_c[k] != '/' &&
182:             expr_c[k] != ' ' && expr_c[k] != '\0' ) {
183:             ++k;
184:         }
185:         strncpy(n_c, expr_c+i, k-i);
186:         n_c[k-i] = '\0';
187:         expr[j].value = static_cast<float>( atoi(n_c) );
188:         if (negative_flag)
189:             expr[j].value = -expr[j].value;
190:         ++j;
191:         i = k - 1;
192:         sign_flag = false;
193:         negative_flag = false;
194:     }
195: }
196: return j;
197: }
198:
199: // -----

```