```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cmath>

using namespace std;


// data structure -------------------------------------------------------

struct Data {
    float value;
    char op;
    bool is_op;
};

class stack
{
    public:
    stack () ;
    void push (Data) ;
    Data pop () ;
    bool empty () ;

    private:
    Data array[1000];
    int index;

};

// ************* Write Your Code Here ************* //
stack::stack(){
    index=0;
};
void stack::push(Data data){
    array[index]=data;
    index++;

};
Data stack::pop(){
    index--;
    return array[index];

};
bool stack::empty(){
    bool emp;
    emp=(index==0)?1:0;
    return emp;
};
// pass
// ************************************************ //

// function declarations ------------------------------------------------

```

```cpp
int convert (char [], Data []) ;

// functions for algorithm ------------------------------------------------

float calculate (float a, char op, float b) {
    if (op == '+')
        return a + b;
    else if (op == '-')
        return a - b;
    else if (op == '*')
        return a * b;
    else
        return a / b;
}


void evaluate (stack &stk) {

    // Fetch the partial expression from stk
    stack stk_tmp;
    Data data;
    // ************* Write Your Code Here ************* //
    // pass
    while(!stk.empty())
    {
        data=stk.pop();
        if(data.is_op&&data.op=='('){
            break;
        }
        stk_tmp.push(data);
    }
    // ************************************************ //

    // Evaluate the partial expression
    float sum1 = 0;
    char op_cur = '+';
    float sum2 = stk_tmp.pop().value;
    while ( !stk_tmp.empty() ) {
        char op_nxt = stk_tmp.pop().op;
        if (op_nxt == '+' || op_nxt == '-') {
            sum1 = calculate(sum1, op_cur, sum2);
            op_cur = op_nxt;
            sum2 = stk_tmp.pop().value;
        }
        else
            sum2 = calculate(sum2, op_nxt, stk_tmp.pop().value);
    }
    data.value = calculate(sum1, op_cur, sum2);
    data.is_op = false;
    stk.push(data);

}

// main function --------------------------------------------------------
```

```cpp

int main ()
{
    while (true) {
        int a=0;
        // Prompt user to input the expression
        char expr_c[1000];
        cout << "Enter the expression: ";
        cin.getline(expr_c, 1000);

        // Exit if the expression is "exit"
        if ( !strcmp(expr_c, "exit") )
            break;

        // Covert character array to our format
        Data expr[1000];
        int expr_size = convert(expr_c, expr);

        // Evaluate the expression
        stack stk;
        // ************* Write Your Code Here ************* //
        // pass
        for(int i=0;i<expr_size;i++)
        {
            if(expr[i].is_op&&expr[i].op==')')
            {
                evaluate(stk);
            }
            else{
                stk.push(expr[i]);
            }
        }
        evaluate(stk);
        // ************************************************ //

        // Show out the result
        cout << "Result: " << stk.pop().value << endl;

    }
    return 0;
}

// function for converting expression format ----------------------------

int convert (char expr_c[], Data expr[]) {
    int j = 0;
    bool sign_flag = false;
    if (expr_c[0] == '+' || expr_c[0] == '-')
        sign_flag = true;
    for (int i = 0; i < strlen(expr_c); ++i) {
        if (expr_c[i] == ' ')
            continue;
        expr[j].is_op = true;
        static bool negative_flag = false;
        if ( expr_c[i] == '(' || expr_c[i] == ')' || expr_c[i] == '+' ||
             expr_c[i] == '-' || expr_c[i] == '*' || expr_c[i] == '/' ) {
            if ( expr_c[i] == '+' || expr_c[i] == '-' ||
                 expr_c[i] == '*' || expr_c[i] == '/' ) {
                if (sign_flag) {
                    if (expr_c[i] == '-')
                        negative_flag = true;
                    continue;
                }
                sign_flag = true;
            }
            expr[j++].op = expr_c[i];
        }
        else {
            expr[j].is_op = false;
            char n_c[10];
            int k = i;
        while ( expr_c[k] != '(' && expr_c[k] != ')' && expr_c[k] != '+' &&
            expr_c[k] != '-' && expr_c[k] != '*' && expr_c[k] != '/' &&
            expr_c[k] != ' ' &&  expr_c[k] != '\0' ) {
            ++k;
            }
            strncpy(n_c, expr_c+i, k-i);
            n_c[k-i] = '\0';
            expr[j].value = static_cast<float>( atoi(n_c) );
            if (negative_flag)
                expr[j].value = -expr[j].value;
            ++j;
            i = k - 1;
            sign_flag = false;
            negative_flag = false;
        }
    }
    return j;
}

// -----------------------------------------------------------------------
```