```
//                              Calc v1.0

// Программа вычисляет результат арифметических выражений, введенных в консоли.
// Для вывода результата, после введения выражения, нужно ввести знак '=' , и нажать 'Enter'.
// Для работы с десятичными дробями можно пользоваться знаками '.' или ','
// Поддерживается следующие операции: '+' , '-' , '*' , '/' , '^' (возведение в степень).
// Приоритет операций: '^' - высший , '*' , '/' - средний, '+' , '-' - низший
// Поддерживается использование скобок, с произвольной степенью вложенности.
// Под выражением "25(1+1)(-2)(3)5" подразумевается произведение из четырех множителей (знаки '*' можно не ставить)
// Между числами и знаками операций можно использовать пробелы.
// Наберите "\Exit", "\exit" или нажмите Ctrl+z и нажмите Enter для выхода из программы

#include <vector>
#include <iostream>
#include <string> // Подключение заголовков

using std::cin;
using std::cout;
using std::endl;
using std::vector;
using std::string;
using std::istream;
using std::ostream; // Объявления using

typedef string::size_type strsz_t;
typedef string::iterator str_iter;
typedef string::const_iterator str_citer; // Определения пользовательских типов // Подключение заголовков, объявления using, определение пользовательских типов

class Error                                                                     // Класс работы с ошибками
{
public:
    Error() = default;
    Error (const string &str) : err_txt("\t>>> Wrong expression: " + str + " <<<") {}
    string what() { return err_txt; }
private:
    string err_txt = "\t>>> Wrong expression <<<";
};
struct token                                                                    // Структура для хранения лексемы
{
    token(char typ = '\0', double val = 0) : type(typ), value(val) {}
    char type;                                                          // Тип лексемы, 'n' - число, или '+' , '-' , '*' , '/' , '^' , '(' , ')'
    double value;                                                               // Числовое значение лексемы (числа или результата в скобках)
};
typedef vector<token>::iterator vectok_iter;
void print_help()
{
    cout << "\tThe program caclulates the result of the ariphmetic expressions,\n"
         << "\ttyped in the console window.\n"
         << "\tTo print the result of the ariphmetic operation enter an expression,\n"
         << "\ta symbol '=', and press Enter.\n"
         << "\tYou can use either symbol '.' or ',' for decimal point\n"
         << "\tYou can use these operations: '+' , '-' , '*' , '/' , '^' (pow)\n"
         << "\tThe priority of the operations is: '^' - high, '*' , '/' - medium,\n"
```

```
        << "\t'+' , '-' - low.\n"
        << "\tYou can use parentheses with the any nesting level in your\n"
        << "\tariphmetic expressions.\n"
        << "\tYou can use space symbols between numbers, operation symbols\n"
        << "\tand parentheses: 1 + 2 / (3 +7 )\n"
        << "\tYou may not use symbol '*' between a numbers and\n"
        << "\tparentheses: 2+3(1+2)(-1)(3)5\n"
        << "\tEnter '\\Exit', '\\exit' or press Ctrl+Z and Enter to exit the program.\n" << endl;
}
inline strsz_t dot_ix(const string &str)                                    // Нахождение положения точки (десятичного разделителя)
{                                                                           // Может использоваться точка или запятая
    strsz_t ix = 0;
    for (; ix != str.size(); ++ix)
        if (str[ix] == '.' || str[ix] == ',')
            return ix;
    return ix;
}
double str2num(const string &str)                                          // Преобразование строки в число типа double
{                                                                           // Может использоваться точка или запятая
    double ret = 0, exp_multplr = pow(10.0, (dot_ix(str) - 1));             // Вычисление экспоненты
    for (auto c : str) {
        if (c == '.' || c == ',')
            continue;
        ret += exp_multplr*(c - '0');
        exp_multplr /= 10;
    }
    return ret;
}
inline vectok_iter right_pair(vectok_iter iter)                             // Нахождение итератора парной скобки
{                                                                           // Параметр - итератор открывающей скобки '(', возвращается итератор
    ++iter;                                                                 // соответствующей ей закрывающей скобки ')', минуя вложенные скобки
    vector<token>::size_type lvl = 0;                                       // Уровень текущей вложенности скобок, 0 - нет внутренних
    while (iter->type != ')' || lvl) {
        if (iter->type == '(')
            ++lvl;
        else if (iter->type == ')')
            --lvl;
        ++iter;
    }
    return iter;
}
void error_check(const vector<token> &tvec)                                 // Функция обработки смысловых ошибок
{
    if (tvec.size() == 1)
        throw Error("No data");
    int pars = 0;                                                           // Счетчик скобок (реверсивный)
    for (auto iter = tvec.begin(); iter != tvec.end()-1; ++iter)                // Цикл проверки лексем
        switch (iter->type) {
            case '+':                                                       // Контроль операции '+'
                if ((iter + 1)->type != 'n' && (iter + 1)->type != '(')     //
                    throw Error();                                          //
                if (iter == tvec.begin())                                   //
                    break;                                                  //
```

```
            else if ((iter - 1)->type != 'n' && (iter - 1)->type != ')' &&
                (iter - 1)->type != '(')
                    throw Error();
            break;
        case '*': case '/': case '^':
            if (iter == tvec.begin())
                throw Error();
            if ((iter + 1)->type != 'n' && (iter + 1)->type != '(')
                throw Error();
            if ((iter - 1)->type != 'n' && (iter - 1)->type != ')')
                throw Error();
            break;
        case 'n':
            if ((iter + 1)->type == 'n')
                throw Error();
            if (iter == tvec.begin())
                break;
            else if ((iter - 1)->type == 'n')
                    throw Error();
            break;
        case '(':
            ++pars;
            if ((iter + 1)->type == ')')
                throw Error();
            break;
        case ')':
            --pars;
            break;
        default:
            throw Error();
        }
    if (pars != 0)
        throw Error("Error at the parentheses");
}
double expr_calc(vectok_iter beg, vectok_iter end)
{
    if (end - beg == 1)
        return beg->value;
    for (auto iter = beg; iter != end - 1; ++iter)
        if (iter->type == '(') {
            if (iter != beg && (iter - 1)->type == '^')
                continue;
            right_pair(iter)->value = expr_calc(iter + 1, right_pair(iter));
        }
        else if (iter->type == '^') {
            if ((iter + 1)->type == '('
                (iter + 1)->value = expr_calc(iter + 2, right_pair(iter + 1));
            (iter - 1)->value = pow((iter - 1)->value, (iter + 1)->value);
        }
    double sum = 0;
    for (auto iter = beg; iter != end; ++iter) {
        switch (iter->type) {
            case '*':
```

```
//
    //
//

// Контроль операций '*', '/' и '^'
//
//
//
//
//
//

// Контроль числовых значений
    //
//
//
//
//
//

// Подсчет скобок
//
    //
//
//
//
//

// Остальные случаи (например '-')
//

// Контроль скобок
//

// Вычисление значения выражения из вектора лексем,
// beg и end - итераторы на начало и конец (после конца) вектора


// Предварительный цикл обработки скобок и степеней
    // Обработка скобок
// Если скобки - это экспонента, то ее значение вычисляется дальше
// Посчитанное в скобках значение сохраняется в лексеме ')'
    // Внутренние скобки вычисляются в рекурсии

// Обработка степеней
    // Вычисление экспоненты, если экспонента - скобки
// Посчитанное значение сохраняется в лексеме '('
// Вычисление степени. Сохраняется в лексеме слева от '^'

// Переменная для вычисления суммы выражения
// Цикл обработки остальных операций

// Операции '*' , '/' и '^' (степень) -
```

```
            (iter + 1)->value *= (iter - 1)->value;                    // результат каждой операции присваивается правой лексеме
            break;                                                     // (правому операнду)
        case '/':                                                      //
            if ((iter + 1)->value == 0)                                // Проверка деления на ноль
                throw Error("Division by zero!");                      //
            else                                                       //
                (iter + 1)->value = (iter - 1)->value / (iter + 1)->value;  //
            break;                                                     //
        case '^':                                                      //
            (iter + 1)->value = (iter - 1)->value;                     //
            break;
        case '(':                                                      // Перенос значения лексемы '(' в лексему ')'
            right_pair(iter)->value *= iter->value;
            break;
        case '=':                                                      // Операция вывода результата выражения '='
            iter->value = (iter - 1)->value;                               // Просто присваивается значение предыдущей лексемы
            break;
        case '+':                                                      // Операции '+' и '-' (минусы уже должны быть заменены на плюсы
            if (iter != beg)                                           // с инвертированием правого операнда в функции tokens_vec_create() )
                sum += (iter - 1)->value;                              //
            break;
        default:                                                       // Остальные случаи:
            ;                                                          // Ни чего не делать
    }
    iter->type = ' ';                                                  // Очистка выражения внутри скобок после вычисления значения выражения
}
return sum ? sum + (end - 1)->value : (end - 1)->value;                // Результирующее значение выражения
}
void tokens_vec_create(vector<token> &tvec, const string &exprn)       // Создание вектора лексем из строки exprn
{
    string number;                                                     // Строка для преобразования текста в число с плавающей запятой
    tvec = {};                                                         // Очистка вектора лексем (для повторного использования)
    for (auto iter = exprn.begin(); iter != exprn.end(); ++iter) {     // Главный цикл
        if (!number.empty())                                           // Обработка числовых лексем
            if (isdigit(*iter) || *iter == '.' || *iter == ',')        // Продолжается считывание числовой лексемы
                number += *iter;                                       //
            else
                tvec.push_back(token('n', str2num(number))), number = "";  // Добавление новой числовой лексемы в вектор
        else
            if (isdigit(*iter))                                        // Новая числовая лексема
                number = *iter;
        if (tvec.size() > 1 && (tvec.end() - 2)->type == '-')          // Замена минусов плюсами и инвертирование правых операндов
            (tvec.end() - 2)->type = '+',                              //
                (tvec.end() - 1)->value = -(tvec.end() - 1)->value;
        if (*iter == '(' && !tvec.empty())                             //
            if ((tvec.end() - 1)->type == 'n' || (tvec.end() - 1)->type == ')')  // Вставка символа '*' между числом и '(' или между ')' и '('
                tvec.push_back(token('*', 1));                         //
        if (isdigit(*iter) && !tvec.empty() && (tvec.end() - 1)->type == ')')  // Вставка символа '*' между ')' и числом
            tvec.push_back(token('*', 1));
        switch (*iter) {                                               //
            case '+':case '-': case '*': case '/':                     // Добавление остальных лексем - операций, скобок и символа '='
            case '^': case '(': case ')':case '=':                     //
                tvec.push_back(token(*iter, 1));                       //
```

```
                break;
            case '.': case ',':
                if (number.empty())
                    throw Error();
                break;
            case ' ':
                break;
            default:                                                    //
                if (!isdigit(*iter))                                    // Обработка текстовых ошибок - не поддерживаемые символы
                    throw Error("Unsupported symbols");                 //
        }
    }
    error_check(tvec);                                                  // Обработка ошибок - смысловые ошибки
}

int main()
{
    cout << "\t\tCalc v1.0\n\tType '\\help' to print help\n" << endl;
    string expression;                                                  // Переменная для хранения введенного выражения
    vector<token> tokens_vec;                                           // Вектор лексем
    while (getline(cin, expression)) {                                  // Главный цикл программы
        if (expression.empty())
            continue;
        if (expression == "\\exit" || expression == "\\Exit")           // Выход
            break;
        else if (expression == "\\help")                                //
            print_help();                                               // Вывод справки
        try {
            if (expression[expression.size() - 1] == '=')               // Обработка выражения
                tokens_vec_create(tokens_vec, expression), cout << "\t\t\t"     //
                << expr_calc(tokens_vec.begin(), tokens_vec.end()) << endl;
            // cout << endl;                                            //
            // for (auto tok : tokens_vec)                              // Отладка
            // cout << tok.type << " : " << tok.value << endl;          //
        }
        catch (Error err) {                                             // Обработка ошибок
            cout << err.what() << endl;                                 //
            continue;                                                   //
        }
    }
    cout << "\n\n" << endl;
    return 0;
}
```