МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: МОиСКТ**

Направление подготовки: Прикладная математика и информатика

Профиль подготовки: Общий профиль

**ОТЧЕТ**

по лабораторной работе

на тему:

**Транслятор арифметических выражений**

**Выполнил:** студент группы 3823Б1ПМ1 Носков И.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Подпись

**Научный руководитель:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Лабинская М.А.

Подпись

Нижний Новгород  
2024

**Оглавление:**

Введение………………………………………………………………………………………….3

I…………………………………………………………………………………………..………..4

II…………………………………………………………………………………….……………..4

III…………………………………………………………………………………….……………4

Заключение…………….………………………………………………………………………..11

Литература………….…………………………………………………………………………...12

**Введение**

В эпоху стремительного развития информационных технологий и программирования важность эффективной обработки данных и выполнения вычислений возрастает как никогда. Арифметические выражения, представляющие собой комбинацию чисел, операторов и скобок, играют ключевую роль в математических вычислениях и программировании. Трансляция арифметических выражений — это процесс преобразования выражений из одной формы представления в другую, что позволяет компьютерам интерпретировать и выполнять математические операции.

1. **Постановка учебно-практической задачи**

*Формулировка задачи:*

Требуется разработать транслятор арифметических выражений, который

будет выполнять следующие функции:

* **Парсинг входного выражения**
* **Преобразование в** постфиксную форму
* **Выполнение вычислений**
* Возможность расширения функционала транслятора

*Исходные данные:*

Expression – строка арифметическое выражение;

*Требуемый результат:*

1. Трансляция арифметического выражение;
2. Вычисление арифметического выражения;

Контрольный пример:

Пример №1. Введем строку 2+7\*(27/3-8.8)+0.5. Рис.1



Рис. 1 Пример №1 – корректное выражение

Получаемся верный ответ = 3.9

Пример №2. Теперь введем другую строку, только так, чтобы в выражение неявно присутствовало деление на ноль. Рис.2



Рис. 2 Пример № 2 – некорректное выражение

1. **Руководство пользователя**

Для выполнения лабораторной работы по использованию транслятора арифметических выражений, необходимо запустить файл **sample.exe.** При запуске появится надпись “Введите арифметическое выражение – “ после чего необходимо ввести арифметическое выражение, которое нужно вычислить. При вводе необходимо использовать только десятичные числа, знаки сложение, вычитание, умножение и деление, так же любые виды скобок, при этом можно разделять операнды пробелы, для удобства ввода.

После ввода строки, если арифметическое выражение корректно, то выведется результат, иначе будет выведено сообщение об ошибке.

1. **Руководство программиста**

**Описание алгоритмов:**

Основная функция вычисление арифметического выражения разбита на несколько стадий:

Лексический анализ, синтаксический анализ, преобразование в обратную польскую запись и вычисление результата.

1. Лексический анализ

Самый первый шаг для работы с арифметическим выражением, это его разбор. В программе реализованы 4 вида лексем: десятичные числа, операции («+» «-» «\*» «/»), открывающая скобка и закрывающая скобка. В алгоритме идем по циклу, и анализируем каждый компонент строки, введенной пользователем. Заполняем вектор указателей на лексемы, который является полем класс транслятора. Также есть возможность добавления новых поддерживаемых функций транслятора, путем добавления их в лексический анализ

void lexical\_analysis() { //Перевод строки в набор лексем

int number\_status = 0; //Флаг для накопления числа

string n; //Строка для накопления числа

for (size\_t i = 0; i < expression.size(); i++) {

if (number\_status == 0) {

if (expression[i] == '(' || expression[i] == '{' || expression[i] == '[')

terms.push\_back(new Open\_Bracket);

if (expression[i] == ')' || expression[i] == '}' || expression[i] == ']')

terms.push\_back(new Close\_Bracket);

if (expression[i] == '+' || expression[i] == '-' || expression[i] == '\*' || expression[i] == '/')

terms.push\_back(new Operation(expression[i]));

if (expression[i] == '0' || expression[i] == '1' || expression[i] == '2' || expression[i] == '3' || expression[i] == '4' || expression[i] == '5' || expression[i] == '6' || expression[i] == '7' || expression[i] == '8' || expression[i] == '9') {

number\_status++;

n += expression[i];

}

}

else {

if (expression[i] == '.' || expression[i] == '0' || expression[i] == '1' || expression[i] == '2' || expression[i] == '3' || expression[i] == '4' || expression[i] == '5' || expression[i] == '6' || expression[i] == '7' || expression[i] == '8' || expression[i] == '9')

n += expression[i];

if (expression[i] == ')' || expression[i] == '}' || expression[i] == ']') {

number\_status = 0;

terms.push\_back(new Number(stod(n)));

terms.push\_back(new Close\_Bracket);

n.clear();

}

if (expression[i] == '+' || expression[i] == '-' || expression[i] == '\*' || expression[i] == '/') {

number\_status = 0;

terms.push\_back(new Number(stod(n)));

terms.push\_back(new Operation(expression[i]));

n.clear();

}

if (expression[i] == '(' || expression[i] == '{' || expression[i] == '[')

throw invalid\_argument("Неверный синтаксис"); //Ошибка синтаксиса

}

}

if(number\_status) //Если осталось накопленное число, добавляем

terms.push\_back(new Number(stod(n)));

};

1. Синтаксический анализ

Проверяем строку на корректность, для этого проверяем два условия: во – первых, выполняются ли условия порядка лексем, в основе функции будем использовать конечный автомат. В столбце стоят лексемы на первом месте, а в строчках – лексемы, следующие за ними. Таб. 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Лексема | Операнд | Операция | Открывающая | Закрывающая |
| Операнд | - | + | - | + |
| Операция | + | - | + | - |
| Открывающая | + | - | + | - |
| Закрывающая | - | + | - | + |

Таб. 1

bool sintaksis\_analisys() const { //Анализ лексем на конечном автомате

types pos = nun; //Сохраняем тип лексемы

if (terms[0]->get\_type() == operation)

return 0; //Первой лексемой не может быть операнд

if (terms[0]->get\_type() == numbers)

pos = numbers;

if (terms[0]->get\_type() == open\_bracket)

pos = open\_bracket;

if (terms[0]->get\_type() == close\_bracket)

return 0; //Первой лексемой не может быть закрывающая скобка

for (size\_t i = 1; i < terms.size(); i++) {

switch (terms[i]->get\_type()) {

case operation:

if (pos == operation || pos == open\_bracket) return 0;

else pos = operation;

break;

case numbers:

if (pos == numbers || pos == close\_bracket) return 0;

else pos = numbers;

break;

case open\_bracket:

if (pos == numbers || pos == close\_bracket) return 0;

else pos = open\_bracket;

break;

case close\_bracket:

if (pos == operation || pos == open\_bracket) return 0;

else pos = close\_bracket;

break;

default:

break;

}

}

return true;

};

Во – вторых нужно проверить правильность расставленных скобок, если они есть, то есть что для каждой открывающейся соответствует такая же, но закрывающая. Так же проверить их последовательность.

bool checking\_brackets() const{ //Проверка скобок

Mstack<char> stack; //Сохраняем скобки в стек

for (size\_t i = 0; i < expression.length(); i++) {

if ((expression[i] == '(') || (expression[i] == '[') || (expression[i] == '{'))

stack.push(expression[i]);

if (expression[i] == ')') {

if (stack.empty()) return 0;

if (stack.top() != '(')

return 0;

stack.pop();

}

if (expression[i] == ']') {

if (stack.empty()) return 0;

if (stack.top() != '[')

return 0;

stack.pop();

}

if (expression[i] == '}') {

if (stack.empty()) return 0;

if (stack.top() != '{')

return 0;

stack.pop();

}

}

if (stack.size()) return 0; //Не для каждой скобки есть пара

return 1;

};

Оба метода помечены как const, так как оба не меняют полей класса, а только проходят в цикле по полям транслятора.

1. Перевод в обратную польскую запись

Преобразуем набор лексем в обратную польскую запись, то есть расставим операции, избавившись от любых скобок. Идем в цикле по набору лексем, если лексема операнд, то добавляем ее в вектор, если операция и стек не пуст, то пока приоритет новой операции меньше или равен приоритету операции на вершине стека, то добавляем в вектор операции из стека, затем пишем текущую операцию в стек. Если открывающая скобка, то пишем в стек, а если закрывающая, то пока не найдем открывающую в стеке, копируем в вектор.

В конце добавляем оставшиеся операции из стека.

void sort\_term() { //Перевод в обратную польскую запись

Mstack<Term\*> stack; //Стек для хранения операций

for (size\_t i = 0; i < terms.size(); i++) {

if (terms[i]->get\_type() == numbers)

sort\_terms.push\_back(terms[i]);

if (terms[i]->get\_type() == operation) {

if (stack.empty())

stack.push(terms[i]);

else { //Пока приоритет новой операции меньше или равен, пишем в вектор

while (!stack.empty() && (stack.top()->get\_type() == operation) && (((Operation\*)(stack.top()))->get\_priority() >= ((Operation\*)(terms[i]))->get\_priority())) {

sort\_terms.push\_back(stack.top());

stack.pop();

}

stack.push(terms[i]);

}

}

if (terms[i]->get\_type() == open\_bracket) {

stack.push(terms[i]);

}

if (terms[i]->get\_type() == close\_bracket) {

while (stack.top()->get\_type() != open\_bracket) {

sort\_terms.push\_back(stack.top());

stack.pop();

}

stack.pop();

}

}

while (!stack.empty()) { //Добавляем оставшиеся операции

sort\_terms.push\_back(stack.top());

stack.pop();

}

};

1. Вычисление результата

С помощью обратной польской записи, производим вычисление результата арифметического выражения.

Если лексема операнд, то добавляем ее в стек, если же лексема – операция, то загружаем из стека первый элемент – это будет правый операнд и сразу же удаляем его из стека, затем берем новый элемент – это будет второй операнд. Определяем значение операции и проводим вычисление, которое кладем на вершину стека.

Так как существует шанс неявного деления на ноль, будем бросать исключение.

double computing() { //Вычисление выражения

Mstack<Term\*> stack; //Стек для хранения лексем

double left\_op, right\_op; //Левый и правый операнды

for (size\_t i = 0; i < sort\_terms.size(); i++) {

if (sort\_terms[i]->get\_type() == numbers)

stack.push(sort\_terms[i]);

if (sort\_terms[i]->get\_type() == operation) {

right\_op = ((Number\*)(stack.top()))->get\_value();

stack.pop();

left\_op = ((Number\*)(stack.top()))->get\_value();

stack.pop();

if (((Operation\*)sort\_terms[i])->get\_operation() == '+')

stack.push(new Number(left\_op + right\_op));

if (((Operation\*)sort\_terms[i])->get\_operation() == '-')

stack.push(new Number(left\_op - right\_op));

if (((Operation\*)sort\_terms[i])->get\_operation() == '\*')

stack.push(new Number(left\_op \* right\_op));

if (((Operation\*)sort\_terms[i])->get\_operation() == '/')

if (right\_op != 0)

stack.push(new Number(left\_op / right\_op));

else //Проверка деления на ноль

throw invalid\_argument("Деление на ноль!");

}

}

return ((Number\*)stack.top())->get\_value(); //Возвращаем ответ

};

1. Итоговая функция

Комбинация функций с 1 – 4 образует итоговую функция класса транслятора. Сначала делаем лексический анализ, затем проверяем на ошибки синтаксиса, если выражение корректно, сортируем вектор указателей и считаем результат, будем выбрасываться исключение.

double translation() { //Итоговая функция вычисления арифметического выражения

lexical\_analysis();

if (!sintaksis\_analisys() || !checking\_brackets())

throw invalid\_argument("Неверный синтаксис"); //Ошибка синтаксиса

sort\_term();

return computing();

};

**Описание структур данных:**

1. Класс лексема

Общая структура лексемы, в private части имеет единственное поле – это тип лексемы. От него наследуются другие структуры, такие как вещественное число, операция, открывающая и закрывающая скобки.

class Term {

private:

types type;

public:

Term(types type) : type(type) {};

types get\_type() const noexcept {

return type;

}

};

//Класс число

class Number : public Term {

private:

double value;

public:

Number(double value): value(value), Term(numbers) {};

double get\_value() const noexcept {

return value;

};

};

//Класс операция

class Operation : public Term {

private:

int priority;

char oper;

public:

Operation(char op = ' ') : oper(op), Term(operation), priority(0) {

if (op == '\*') priority = 2;

if (op == '/') priority = 2;

if (op == '+') priority = 1;

if (op == '-') priority = 1;

};

char get\_operation() const noexcept {

return oper;

};

int get\_priority() const noexcept {

return priority;

};

};

//Класс открывающая скобка

class Open\_Bracket : public Term {

private:

char bracket;

public:

Open\_Bracket(): Term(open\_bracket), bracket('(') {};

};

//Класс закрывающая скобка

class Close\_Bracket : public Term {

private:

char bracket;

public:

Close\_Bracket(): Term(close\_bracket), bracket(')') {};

};

1. Класс транслятор

Главная структура, работающая с арифметическим выражением. В private части имеет поля: строка – арифметическое выражение, два вектора указателей на лексемы( первый заполняется при лексической анализе арифметического выражения, второй получается переводов в обратную польскую запись)

Методы класса – лексический анализ, синтаксический анализ, преобразование в обратную польскую запись и вычисление результата. Все эти методы объединены в один метод translation(), который позволяется полностью проанализировать арифметическое выражение и вернуть ответ, если выражение корректно.

class translator {

private:

string expression; //Строка выражение

Mvector<Term\*> terms; //Вектор лексем

Mvector<Term\*> sort\_terms; //Вектор лексем в обратной польской записи

public:

translator(string str) : expression(str) {};

void lexical\_analysis() {};

bool sintaksis\_analisys() const {};

bool checking\_brackets() const {};

void sort\_term() {};

double computing() {};

double translation() {};

};

**Заключение**

В этой лабораторной работе был разработан транслятор арифметических выражений, который выполняет следующие функции: **парсинг входного выражения, преобразование в** постфиксную форму, **выполнение вычислений. Также существует** возможность расширения функционала транслятора.

**Литература**

* Кнут Д. Э. Искусство программирования для ЭВМ. Том 2. Получисленные алгоритмы — М.: Мир, 1977. — 724 с.