

ВВЕДЕНИЕ

Арифметическое выражение представляет собой такое выражение, в котором объекты действуют в качестве операндов и выполняются арифметические операции над ними.

В инфиксной форме записи операторы находятся между operandами, и порядок выполнения операций определяется приоритетом операторов и расстановкой скобок. В отличие от этого, постфиксная форма записи не содержит скобок, а операторы располагаются справа от аргументов.

Обратная польская нотация была разработана австралийским ученым Чарльзом Хэмблином в середине 1950-х годов на основе польской нотации, предложенной польским математиком Яном Лукасевичем в 1920 году. Постфиксная форма записи является основой для организации вычислений арифметических выражений.

Ученый Эдсгер Дейкстра предложил алгоритм который позволяет преобразовывать выражения из инфиксной формы в постфиксную форму.

В данной лабораторной работе требуется изучить и реализовать алгоритм сортировочной станции – алгоритм Эдсгера Дейкстры преобразования инфиксной записи в постфиксную.

Для достижения поставленных целей лабораторной работы необходимо выполнить следующие задачи:

- рассмотреть и изучить алгоритм сортировочной станции;
- реализовать алгоритм;
- составить блок-схему алгоритма;
- рассчитать трудоемкость алгоритма;
- экспериментально проверить работу алгоритма;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

Математические выражения в компьютере могут быть записаны в инфиксной, постфиксной или префиксной форме. В инфиксной форме операнды располагаются между операторами, что может вызывать неоднозначность порядка выполнения операций. Однако в постфиксной и префиксной формах операции следуют непосредственно после соответствующих операндов, что позволяет компьютеру вычислять выражение последовательно без неоднозначности порядка операций [3].

Для решения этой задачи был придуман алгоритм преобразования в постфиксную форму записи нидерландским ученым Эдсгером Дейкстрой. Алгоритм сортировочной станции предложен Э. Дейкстрой и назван им так, потому что напоминает процесс сортировки на железнодорожной грузовой сортировочной станции.

Алгоритм работает при помощи стека: для преобразования в обратную польскую нотацию используется входная и выходная строки, а также стек для хранения операторов, ещё не добавленных в выходную очередь. При преобразовании алгоритм считывает 1 символ и производит действия, зависящие от данного символа [5].

Поскольку операторов в языке программирования C++ для данной лабораторной работы много, ограничимся лишь следующий списком:

- Унарный минус;
- Сложение;
- Вычитание;
- Деление;
- Умножение;
- Нахождение остатка.

Таким образом, имея 2 уровня приоритета операций, а также стек, который необходимо реализовать, можно перейти к этапу сборки программы.

Вывод

На данном этапе была изучена история алгоритма сортировочной станции. Кратко описаны идея алгоритма и принцип его работы.

2 Конструкторская часть

2.1 Разработка алгоритма

В соответствии с аналитической частью далее можно составить описательную схему алгоритма сортировочной станции:

Пока не все символы обработаны:

1. Прочитать символ.
2. Если символ — операнд, то добавить его в очередь вывода.
3. Если символ — операция, то поместить его в стек.
4. Если символ — оператор op1, то:
 - a. Пока присутствует на вершине стека символ оператор op2, чей приоритет выше или равен приоритету op1, и при равенстве приоритетов op1 является левоассоциативным:
 - i. Переложить op2 из стека в выходную очередь;
 - b. Положить op1 в стек.
 5. Если символ — открывающая скобка, то положить его в стек.
 6. Если символ — закрывающая скобка:
 - a. Пока символ на вершине стека не открывающая скобка
 - i. Переложить оператор из стека в выходную очередь.
 - ii. Если стек закончился до того, как был встречен символ открывающая скобка, то в выражении пропущена скобка.
 - b. Выкинуть открывающую скобку из стека, но не добавлять в очередь вывода.
 - c. Если символ на вершине стека — функция, переложить её в выходную очередь.

Если больше не осталось символов на входе:

1. Пока есть символы операторы в стеке:
 - a. Если символ оператор на вершине стека — открывающая скобка, то в выражении пропущена скобка.
 - b. Переложить оператор из стека в выходную очередь.

Конец алгоритма.

2.2 Трудоемкость алгоритмов

Для последующего вычисления трудоемкости алгоритмов необходимо ввести модель вычислений:

- 1) Пусть: +, -, /, %, ==, !=, <, >, <=, >=, [], *, ++, -- – трудоемкость 1;
- 2) Трудоемкость оператора выбора if условие then A else B рассчитывается как на рисунке 4;

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}$$

Рисунок 1 - Трудоемкость оператора выбора if условие then A else B

- 3) Трудоемкость цикла for рассчитывается как на рисунке 5;

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$$

Рисунок 2 - Трудоемкость оператора for

- 4) Трудоемкость вызова функции стека равна 0.

Произведём вычисление трудоёмкости алгоритма. В таблице 1 приведены трудоемкости для лучшего и худшего случаев.

Таблица 1 – Расчёт трудоемкости алгоритма сортировочной станции

Трудоемкость алгоритма сортировочной станции	
Худший случай	Лучший случай
$O(n)$	$O(n)$

Вывод

С учетом теоретических данных, полученных из аналитического раздела, разработана описательная схема алгоритма Дейкстры для получения постфиксного выражения. Кроме того, проведена теоретическая оценка трудоемкости алгоритма.

3 Экспериментальная часть

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся инфиксное выражение из целых чисел;
- на выходе — постфиксное выражение из целых чисел, результат арифметического выражения.

3.2 Технические характеристики

Технические характеристики устройства, на котором происходило тестирование:

- Операционная система: Windows 10 Pro 64 бит
- Память: 8 ГБ
- Процессор: Intel Core i5-1135G7 2.4 ГГц

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

3.3 Тестирование программы

В таблице 2 приведены функциональные тесты для программы, реализующей алгоритм сортировочной станции – все тесты были пройдены успешно.

Таблица 2 – Тестирование программы

Входной массив	Выходной массив	Ожидаемый результат
$3+3^*4-6+3^*(4+6/3)$	$334^*+6-3463/+^*$ 27	$334^*+6-3463/+^*$ 27
$3^*7/3$	$37^*3/$ 7	$37^*3/$ 7
$24-10^*2-3+7^*1$	24102^*-3-71^* 5	24102^*-3-71^* 5
{Пустая строка}	Ошибка	Ошибка
0+1	01+	01+
0	0	0

Вывод

В рамках данной части был реализован и протестирован алгоритм Дейкстры для получения из инфиксного выражения постфиксной формы записи.

Ожидаемые результаты совпали с выходными данными – тесты были проведены успешно.

Стек был реализован самостоятельно с помощью связного списка. Для обработки ошибок были рассмотрены неисчерпывающие себя сценарии такие как ввод пустой строки, ввод неверного символа и другие.

Алгоритм сортировочной станции, реализованный в этом разделе, успешно преобразует арифметическое выражение в инфиксной форме в выражение постфиксной формы, а также вычисляет значение данного выражения.

ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы были проведены анализ, подробное изучение, реализация и тестирование алгоритма Дейкстры для получения постфиксного арифметического выражения. Выполнены следующие задачи:

- рассмотрен и изучен алгоритм сортировочной станции;
- реализован данный алгоритм;
- составлена блок-схема алгоритма;
- рассчитана трудоемкость алгоритма;
- экспериментально проверена работа алгоритма;
- на основании проделанной работы сделаны выводы.

ПРИЛОЖЕНИЕ А

Листинг программы

Листинг 1 – Алгоритм Дейкстры для постфиксного выражения

```
#include <iostream>
#include <string>

using namespace std;

//Структура узла стека
struct StackNode {
    char data;
    StackNode* next;

    StackNode(char value) : data(value), next(nullptr) {}
};

//Класс стека
class Stack {
private:
    StackNode* top;

public:
    Stack() : top(nullptr) {}

    ~Stack() {
        while (!isEmpty()) {
            pop();
        }
    }

    //Проверка, является ли стек пустым
    bool isEmpty() {
        return top == nullptr;
    }

    //Добавление элемента в стек
    void push(char value) {
        StackNode* newNode = new StackNode(value);
        newNode->next = top;
        top = newNode;
    }

    //Извлечение верхнего элемента из стека
    char pop() {
        if (isEmpty()) {
            cout << "STACK IS EMPTY" << endl;
            return '\0'; //Возвращаем значение по умолчанию в случае ошибки
        }
        StackNode* temp = top;
        char data = temp->data;
        top = top->next;
        delete temp;
        return data;
    }
}
```

```

}

//Получение верхнего элемента стека без удаления
char peek() {
    if (isEmpty()) {
        cout << "STACK IS EMPTY" << endl;
        return '\0'; //Возвращаем значение по умолчанию в случае ошибки
    }
    return top->data;
}

//Функция, проверяющая, является ли символ оператором
bool isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/' || c == '%';
}

//Функция, возвращающая приоритет оператора
int getPriority(char c) {
    if (c == '+' || c == '-')
        return 1;
    if (c == '*' || c == '/' || c == '%')
        return 2;
    return 0;
}

//Функция, преобразующая инфиксное выражение в постфиксное
string infixToPostfix(string infix) {
    Stack operatorStack;
    string postfix;

    for (char c : infix) {
        if (isalnum(c)) {
            postfix += c;
        } else if (c == '(') {
            operatorStack.push(c);
        } else if (c == ')') {
            while (!operatorStack.isEmpty() && operatorStack.peek() != '(') {
                postfix += operatorStack.pop();
            }
            if (operatorStack.isEmpty()) {
                cout << "ERROR: NOT MATCHING BRACKETS" << endl;
                return "";
            }
            operatorStack.pop(); //Удаляем открывающую скобку
        } else if (isOperator(c)) {
            while (!operatorStack.isEmpty() && operatorStack.peek() != '(' &&
                   getPriority(operatorStack.peek()) >= getPriority(c)) {
                postfix += operatorStack.pop();
            }
            operatorStack.push(c);
        } else {
            cout << "ERROR: WRONG SYMBOL" << endl;
            return "";
        }
    }

    while (!operatorStack.isEmpty()) {
        if (operatorStack.peek() == '(') {
            cout << "ERROR: NOT MATCHING BRACKETS" << endl;
            return "";
        }
        postfix += operatorStack.pop();
    }
}

```

```

    }

    return postfix;
}

//Функция, вычисляющая результат постфиксного выражения
double evaluatePostfix(string postfix) {
    Stack operandStack;

    for (char c : postfix) {
        if (isdigit(c)) {
            operandStack.push(c - '0');
        } else if (isOperator(c)) {
            double operand2 = operandStack.pop();
            double operand1 = operandStack.pop();

            double result;
            switch (c) {
                case '+':
                    result = operand1 + operand2;
                    break;
                case '-':
                    result = operand1 - operand2;
                    break;
                case '*':
                    result = operand1 * operand2;
                    break;
                case '/':
                    if (operand2 == 0) {
                        cout << "ERROR: DIVIDING BY ZERO" << endl;
                        return 0;
                    }
                    result = operand1 / operand2;
                    break;
                case '%':
                    if (operand2 == 0) {
                        cout << "ERROR: DIVIDING BY ZERO" << endl;
                        return 0;
                    }
                    result = operand1 % operand2;
                    break;
            }
            operandStack.push(result);
        } else {
            cout << "ERROR: WRONG SYMBOL" << endl;
            return 0;
        }
    }

    if (operandStack.isEmpty()) {
        cout << "ERROR: EPRTY EXPRESSION" << endl;
        return 0;
    }

    return operandStack.pop();
}

int main() {
    string infixExpression;
    cout << "Enter an infix expression: ";
    getline(cin, infixExpression);
}

```

```
if (infixExpression.empty()) {
    cout << "ERROR: EMPTY STRING!" << endl;
    return 0;
}

string postfixExpression = infixToPostfix(infixExpression);

if (postfixExpression.empty()) {
    return 0;
}

cout << "Postfix expression: " << postfixExpression << endl;

double result = evaluatePostfix(postfixExpression);
cout << "Expression result: " << result << endl;

return 0;
}
```