Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №2**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Перевод исходной программы в обратную польскую запись»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. В. Максюта

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. М. Вишняков

Краснодар

2024

**СОДЕРЖАНИЕ**

[1 Вариант задания 3](#_Toc103025612)

[2 Понятие обратной польской записи 3](#_Toc103025613)

[3 Алгоритм Дейкстры 4](#_Toc103025614)

[4 Перевод операторов цикла в ОПЗ 7](#_Toc103025615)

[5 Результаты экспериментов 9](#_Toc103025616)

**1 Вариант задания**

Вариант задания представляет собой пару: входной язык и выходной язык (таблица 1).

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 5 | Python | JavaScript |

Разработать программу для перевода закодированного текста исходной программы в обратную польскую запись.

Программа получает на входе файл – результат лексического анализа и строит обратную польскую запись исходной программы.

Отчет по работе должен содержать полное описание алгоритма Дейкстры: таблицу приоритетов операторов и операций, а также алгоритм работы со стеком. Листинг программы и комментарии к нему, пример.

**2 Понятие обратной польской записи**

Обратная польская запись (ОПЗ) – представляет собой одну из форм записи выражений и операторов, отличительной особенностью которой является расположение аргументов (операндов) перед операцией (оператором).

Например, выражение, записанное в обычной скобочной записи,

(a+d)/c+b\*(e+d),

в ОПЗ имеет следующее представление:

ad+c/bed+\*+.

Обратная польская запись получила широкое распространение благодаря своему основному преимуществу ОПЗ может быть вычислена за один просмотр цепочки слева направо, который часто называют проходом.

**3 Алгоритм Дейкстры**

Исследованию формальных способов преобразования арифметических и логических выражений в ОПЗ посвящены многочисленные исследования, однако в практике системного программирования наибольшее распространение получили способы преобразования на основе алгоритма Дейкстры.

Суть алгоритма Дейкстры можно представить следующим рисунком:

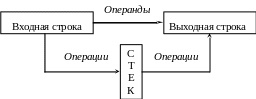


Рисунок 1 – Суть алгоритма Дейкстры

Из этого рисунка следует, что на вход алгоритма посимвольно поступает исходное выражение. Операнды исходного выражения пропускаются на выход и формируют так же посимвольно выходную строку. Операции обрабатываются по определенным правилам на основе стека.

Для реализации такой обработки известное в системном программировании понятие стека используется также в алгоритме Дейкстры для размещения в нем операций. При этом предварительно каждой операции приписывается свой приоритет на основе таблицы приоритетов, которая приведена ниже (таблица 2).

Таблица 2 – Таблица приоритетов

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| (  for  if  while  [  АЭМ  Ф | 0 |
| )  ,  do  else  elif  ] | 1 |
| = | 2 |
| or | 3 |
| and | 4 |
| not | 5 |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Входной элемент | Приоритет |
| <  <=  !=  =  >  >= | 6 |
| +  -  +=  -=  \*=  /=  \*\*=  %= | 7 |
| \*  /  % | 8 |
| return | 9 |

**4 Перевод операторов цикла в ОПЗ**

Обработка оператора цикла с предусловием WHILE выражение DO оператор;:

1. Символ WHILE из входной строки заносится в стек. В стеке к символу WHILE добавляется рабочая метка Mi и после этого в выходную строку записывается часть Mi:.
2. Символ DO выталкивает в выходную строку все операции из стека до ближайшего WHILE Mi. В стеке к WHILE Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается часть Mi+1 УПЛ.
3. Символ ‘;’ указывает на конец оператора цикла с предусловием и выталкивает из стека все символы до ближайшего WHILE Mi Mi+1, при этом сам WHILE уничтожается, а в выходную строку помещается Mi БП Mi+1:.

Обработка оператора цикла с постусловием REPEAT оператор UNTIL выражение; можно заменить последовательностью операторов

1. Символ ‘;’ указывает на конец оператора цикла с постусловием и выталкивают из стека все символы до ближайшего REPEAT Mi, при этом сам REPEAT уничтожается, а в выходную строку помещаются Mi+1 УПЛ Mi БП Mi+1:.

Обработка оператора цикла с счетчиком FOR переменная = значение1 TO значение2 DO оператор;:

1. FOR переменная из входной строки заносится в стек.
2. Символ TO выталкивает в выходную строку все операции из стека до ближайшего FOR переменная. В стеке к FOR переменная добавляется рабочая метка Mi и после этого в выходную строку записывается Mi: переменная.
3. Символ DO выталкивает в выходную строку все операции из стека до ближайшего FOR переменная Mi. В стеке к FOR переменная Mi добавляется рабочая метка Mi+1 и после этого в выходную строку записывается <= Mi+1 УПЛ.
4. Символы ‘;’ и END указывают на конец оператора цикла с счетчиком и выталкивают из стека все символы до ближайшего FOR переменная Mi Mi+1, при этом сам FOR уничтожается, а в выходную строку помещается метка переменная переменная 1 + := Mi БП Mi+1:.

**5 Результаты экспериментов**

Пример входных данных (текста программы):

def calculate\_statistics(numbers):

    total = 0

    count = len(numbers)

    for number in numbers:

        total += number

    if count > 0:

        average = total / count

    else:

        average = 0

    return total, average, count

def main():

    numbers = [10, 20, 30, 40, 50]

    total, average, count = calculate\_statistics(numbers)

    print("Сумма чисел:", total)

    print("Среднее значение:", average)

    print("Количество элементов:", count)

    if total > 100:

        print("Сумма больше 100")

    elif total == 100:

        print("Сумма равна 100")

    else:

        print("Сумма меньше 100")

    demonstrate\_loops()

def demonstrate\_loops():

    array = [1, 2, 3, 4, 5]

    found = False

    target = 3

    for index, value in enumerate(array):

        if value == target:

            found = True

            break

    if found:

        print(f"Значение {target} найдено в массиве на позиции {index}.")

    else:

        print(f"Значение {target} не найдено в массиве.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Пример выходного файла:

def calculate\_statistics numbers : total 0 count len numbers

for number in numbers : total + number if count 0 : average > total count else : average / 0 return total ,

average , count def main : numbers [ 10 , 20 , 30 , 40 , 50 ] total , average , count calculate\_statistics numbers print "Сумма чисел:" ,

total print "Среднее значение:" , average print "Количество элементов:" , count if total 100 : print "Сумма

больше 100" elif total > 100 : print "Сумма равна 100" else : print "Сумма меньше 100"

demonstrate\_loops def demonstrate\_loops : array [ 1 , 2 , 3 , 4 , 5 ] found

False target 3 for index , value in enumerate array : if value target

: found True break if found : print f "Значение {target} найдено в массиве на позиции {index}." else :

print f "Значение {target} не найдено в массиве." if \_\_name\_\_ "\_\_main\_\_

" : main = = = = = = = = = = = = = = = = =

Код программы:

class ShuntingYard {

    constructor() {

        this.outputQueue = [];

        this.operatorStack = [];

        this.operators = {

            '+': { precedence: 7, associativity: 'L' },

            '-': { precedence: 7, associativity: 'L' },

            '\*': { precedence: 8, associativity: 'L' },

            '/': { precedence: 8, associativity: 'L' },

            '%': { precedence: 8, associativity: 'L' },

            '^': { precedence: 8, associativity: 'R' },

            '<': { precedence: 6, associativity: 'L' },

            '<=': { precedence: 6, associativity: 'L' },

            '!=': { precedence: 6, associativity: 'L' },

            '==': { precedence: 6, associativity: 'L' },

            '>': { precedence: 6, associativity: 'L' },

            '>=': { precedence: 6, associativity: 'L' },

            '=': { precedence: 2, associativity: 'R' },

            'or': { precedence: 3, associativity: 'L' },

            'and': { precedence: 4, associativity: 'L' },

            'not': { precedence: 5, associativity: 'R' },

            'return': { precedence: 9, associativity: 'L' }

        };

        this.keywords = new Set(['if', 'else', 'elif', 'for', 'while', 'do', 'return']);

        this.otherTokens = new Set([':', ',', '[', ']', '{', '}', '=>']);

    }

    processToken(token) {

        if (!isNaN(token) || /^[a-zA-Z\_][a-zA-Z0-9\_]\*$/.test(token) || /^['"].\*['"]$/.test(token)) {

            this.outputQueue.push(token);

        } else if (this.operators[token]) {

            while (

                this.operatorStack.length &&

                this.operators[this.operatorStack[this.operatorStack.length - 1]] &&

                (

                    (this.operators[token].associativity === 'L' &&

                    this.operators[token].precedence <= this.operators[this.operatorStack[this.operatorStack.length - 1]].precedence) ||

                    (this.operators[token].associativity === 'R' &&

                    this.operators[token].precedence < this.operators[this.operatorStack[this.operatorStack.length - 1]].precedence)

                )

            ) {

                this.outputQueue.push(this.operatorStack.pop());

            }

            this.operatorStack.push(token);

        } else if (token === '(') {

            this.operatorStack.push(token);

        } else if (token === ')') {

            while (this.operatorStack.length && this.operatorStack[this.operatorStack.length - 1] !== '(') {

                this.outputQueue.push(this.operatorStack.pop());

            }

            this.operatorStack.pop(); *// Удалить '(' из стека*

            if (this.operatorStack.length && /^[a-zA-Z\_][a-zA-Z0-9\_]\*$/.test(this.operatorStack[this.operatorStack.length - 1])) {

                this.outputQueue.push(this.operatorStack.pop()); *// Вставить имя функции в выходную очередь*

            }

        } else if (this.keywords.has(token)) {

            this.outputQueue.push(token);

        } else if (this.otherTokens.has(token)) {

            this.outputQueue.push(token);

        } else {

            throw new Error(`Unknown token: ${token}`);

        }

    }

    finalize() {

        while (this.operatorStack.length) {

            this.outputQueue.push(this.operatorStack.pop());

        }

    }

    toRPN(tokens) {

        tokens.forEach(token => this.processToken(token));

        this.finalize();

        return this.outputQueue;

    }

}

*// Функция для разбора кода на токены*

function tokenize(input) {

    const regex = /\s\*(=>|[-+\*/%^=()<>{}[\],:]|[A-Za-z\_]\w\*|\d+|".\*?"|'.\*?'|.)\s\*/g;

    return input.split(regex).filter(s => !s.match(/^\s\*$/));

}

*// Чтение данных из файла и запуск алгоритма*

const fs = require('fs');

fs.readFile('1.py', 'utf8', (err, data) => {

    if (err) {

        console.error(err);

        return;

    }

    const tokens = tokenize(data);

    const sy = new ShuntingYard();

    const rpn = sy.toRPN(tokens);

    const outputPath = 'output2.txt';

    fs.writeFile(outputPath, rpn.join(' '), (err) => {

        if (err) {

            console.error(err);

            return;

        }

        console.log(`RPN записана в файл ${outputPath}`);

    });

});