

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Новосибирский государственный технический университет»

Кафедра прикладной математики

**ОТЧЕТ ПО ПРАКТИКЕ**

Учебная практика: ознакомительная практика  
(наименование практики в соответствии с учебным планом)

Направление подготовки: 01.03.02 Прикладная математика и информатика

Профиль: Компьютерное моделирование и информационные технологии.

Выполнил:

Студент Коренко Юлия Олеговна  
(Ф.И.О.)

Группа ПМ-15

Факультет ПМИ

Проверил:

Руководитель от НГТУ Вагин Денис Владимирович,  
(Ф.И.О.)

Балл: \_\_\_\_\_, ECTS \_\_\_\_\_,

Оценка \_\_\_\_\_  
«отлично», «хорошо», «удовлетворительно», «неуд.»,  
«зачтено», «не зачтено»

\_\_\_\_\_  
*подпись*

«\_\_» \_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_  
*подпись*

«\_\_» \_\_\_\_\_ 20\_\_ г.

Новосибирск 2024

## Оглавление

1. Цели ознакомительной практики.....	3
2. Теоретическая часть.....	3
3. Тестирование.....	4
4. Текст программ.....	7

## Цель ознакомительной практики

Создание синтаксического анализатора для калькулятора.

## Поставленные задачи

1. Изучение формальных грамматик, ДКА, способов токенизации, построения постфиксной записи.
2. Создание скрипта и калькулятора на языке Python. Верстка сайта, подключение стилей и скрипта.
3. Тестирование и отладка веб-сайта.

## Теоретическая часть

Синтаксический анализ - это процесс проверки синтаксиса входной строки для определения ее структуры и соответствия определенному формальному грамматике. В контексте сайта-калькулятора интегралов синтаксический анализатор будет проверять введенные пользователем выражения на предмет правильности синтаксиса и определять их структуру.

Синтаксический анализатор - это программа, которая реализует формальную грамматику и проверяет введенный пользователем текст на соответствие этой грамматике.

Постфиксная запись, также известная как обратная польская запись, - это нотация, в которой операторы записываются после своих операндов. Например, выражение `'1 + 2'` в постфиксной записи будет записано как `'1 2 +'`.

Реализуемый синтаксический анализатор способен обрабатывать числовые литералы, операции (+, -, /, \*, ^), функции (синус, косинус, натуральный логарифм, возведения экспоненты в степень). Так же учитывается приоритет операций.

Алгоритм составления постфиксной записи

Вход: Инфиксное выражение

Выход: Постфиксная запись

1. Создать пустой стек.
2. Для каждого символа в инфиксном выражении:
  - Если символ является операндом (константа или переменная), добавить его в выходную строку.
  - Если символ является открывающей скобкой, добавить его в стек.
  - Если символ является закрывающей скобкой, выталкивать элементы из стека и добавлять их в выходную строку до тех пор, пока не встретится открывающая скобка. Удалить открывающую скобку из стека.
  - Если символ является оператором:
    - Если стек пуст или содержит только открывающие скобки, добавить оператор в стек.
    - В противном случае, пока стек не пуст и приоритет оператора в стеке выше или равен приоритету текущего оператора:
      - Вытолкнуть оператор из стека и добавить его в выходную строку.
      - Добавить текущий оператор в стек.
3. Вытолкнуть все оставшиеся операторы из стека и добавить их в выходную строку.

Синтаксический анализатор, реализованный на сайте-калькуляторе интегралов, использует метод спуска с рекурсией. Он принимает в качестве входных данных строковое выражение, введенное пользователем, и проверяет его на соответствие формальной грамматике. Если выражение является синтаксически правильным, синтаксический анализатор преобразует его в постфиксную запись. Если выражение синтаксически неправильно, синтаксический анализатор выводит сообщение об ошибке. Автоматизация вычисления выражений в обратной польской нотации основана на использовании стека.

### Тестирование

Калькулятор написан для численного интегрирования методом прямоугольников(левых)  
Постфиксная запись записывается в rpn.txt

Простой тест

$\int$    $dx$

$x_1:$    $x_2:$   Разбиения:

Уравнение:  $x^2*(x+7)$

Разбиений: 100

Шаг: 0.09

Результат: 4754.895524999998

rpn.txt:  $x^2 * x + 7$

## Тест с функциями

$\int$    $dx$

$x_1$ :   $x_2$ :  Разбиения:

Уравнение:  $x^2 \cdot \exp(\sin(x) + 7)$

Разбиений: 100

Шаг: 0.01

Результат: 6663.142234544498

rpn.txt:  $x^2 \cdot \exp(\sin(x) + 7)$

## Тест с неверно заданной подынтегральным выражением

$\int$    $dx$

$x_1$ :   $x_2$ :  Разбиения:

Ошибка! Неверно введено подынтегральное выражение

## Текст программ

nav.php

```
<nav>
    
    <div class="col-1 mt-4">Mega.solver</div>
</nav>
```

index.php

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" />
    <title>Math Solver</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRN+PtmoHDEXuppvndJzQIu9"
crossorigin="anonymous">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap"
rel="stylesheet">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900;1,
100..900&family=Nunito+Sans:ital,opsz,wght@0,6..12,200..1000;1,6..12,200..1000&di
splay=swap" rel="stylesheet">
    <link rel="stylesheet" href="style.css" />
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
</head>
<body>
    <?php include ("nav.php"); ?>
    <div class="container col-15">

        <div class="menu col-3">
            <h1>Задачи</h1>
            <div class = "sidebar">
                <ul>
                    <li>
                        <a href="index.php">Интегралы</a>
                    </li>
                    <li>
                        <a href="matrix.php">Матрицы</a>
                    </li>
                </ul>
            </div>
        </div>
    </div>
```

```

    </div>
</div>

<div class="calc col-8">
    
    <div class="col">
        <input type="text" class="equation" value="x^2+x">
        <button name="letsgo" type="submit" class="btn btn-
primary">Решить</button>
        <span>dx</span>
    </div>
    <div class="param col-1">
        <p>x<sub>1</sub>: </p><input type="text" class="a col-2"
value="1">
        <p>x<sub>2</sub>: </p><input type="text" class="b col-2"
value="10">
        <p>Разбиения: </p><input type="text" class="n col-2" value="100">
    </div>
    <div class="result" hidden>
        <p class="row" id="infix"></p>
        <p class="det row" id="det" hidden></p>
        <p class="step row" id="step" hidden></p>
        <p class="res row" id="res" hidden></p>
    </div>
</div>

<div class="info col-4">
    <h1>Инструкция</h1>
    <p>Возведение в степень задается символом - <span>^</span></p>
    <p>Если ты хочешь посчитать натуральный логарифм - пиши
<span>log()</span></p>
    <p>А возведение экспоненты в степень делается так -
<span>exp()</span></p>
</div>

</div>
<script src="script.js"></script>
<script type="text/javascript">
    $(''.btn-primary').click(function (){
        const equationElement = document.querySelector(".equation");
        const equationValue = equationElement.value;
        const aElement = document.querySelector(".a");
        const aValue = aElement.value;
        const bElement = document.querySelector(".b");
        const bValue = bElement.value;
        const nElement = document.querySelector(".n");
        const nValue = nElement.value;
        var data = {
            equation: equationValue,
            param: aValue + " " + bValue + " " + nValue,

```

```

    });
    console.log(data);
    $.ajax({
        url: 'control.php',
        method: 'post',
        data: data,
        success: function(response){
            const arr = response.split("\n");
            const infix = arr[0];
            const det = arr[1];
            const step = arr[2];
            const res = arr[3];
            if (res == undefined)
            {
                document.getElementById("infix").innerHTML = 'Ошибка! Неверно
введено подынтегральное выражение';
                const str1 = document.querySelector('.det');
                str1.setAttribute('hidden', 'true');
                const str2 = document.querySelector('.step');
                str2.setAttribute('hidden', 'true');
                const str3 = document.querySelector('.res');
                str3.setAttribute('hidden', 'true');
            }
            else
            {
                const str1 = document.querySelector('.det');
                str1.removeAttribute('hidden');
                const str2 = document.querySelector('.step');
                str2.removeAttribute('hidden');
                const str3 = document.querySelector('.res');
                str3.removeAttribute('hidden');
                document.getElementById("infix").innerHTML = 'Уравнение: ' +
infix;

                document.getElementById("det").innerHTML = 'Разбиений: ' + det;
                document.getElementById("step").innerHTML = 'Шаг: ' + step;
                document.getElementById("res").innerHTML = 'Результат: ' + res;
            }
            const result = document.querySelector('.result');
            result.removeAttribute('hidden');
        }
    });
})
</script>
</body>
</html>

control.php
<?php

```



```

$val = $_POST['equation'];
$param = $_POST['param'];
$file = fopen("infix.txt", "w");
fwrite($file, $val);
fclose($file);
$file = fopen("param.txt", "w");
fwrite($file, $param);
fclose($file);
exec('toRPN.exe');
exec('RPN_calc.exe');

$file_res = fopen("res.txt", "r");

$res = fread($file_res, filesize('res.txt'));
echo($val);
echo("\n");
echo($res);

fclose($file_res);

$test = fopen("infix.txt", "r");
$infix = fread($test, filesize('infix.txt'));
fclose($test);

```

RPN\_calc.py

```

import math as m

def toInfix(rpn_string: str):
    stack = []
    func = ''
    for token in rpn_string.split():
        if token == '+':
            a = (stack.pop());
            b = stack.pop();
            tmp = '(' + b + ' + ' + a + ')'
            stack.append(tmp)
        elif token == '-':
            a = stack.pop();
            b = stack.pop();
            tmp = '(' + b + ' - ' + a + ')'
            stack.append(tmp)
        elif token == '/':
            a = stack.pop();
            b = stack.pop();
            tmp = b + ' / ' + a
            stack.append(tmp)
        elif token == '*':
            a = stack.pop();
            b = stack.pop();

```

```

        tmp = b + ' * ' + a
        stack.append(tmp)
    elif token == '^':
        a = stack.pop();
        b = stack.pop();
        tmp = b + ' ** ' + a
        stack.append(tmp)
    elif token == 'exp':
        a = stack.pop();
        tmp = 'exp(' + a + ')'
        stack.append(tmp)
    elif token == 'log':
        a = stack.pop();
        tmp = 'log(' + a + ')'
        stack.append(tmp)
    elif token == 'sqrt':
        a = stack.pop();
        tmp = 'sqrt(' + a + ')'
        stack.append(tmp)
    elif token == 'cos':
        a = stack.pop();
        tmp = 'cos(' + a + ')'
        stack.append(tmp)
    elif token == 'sin':
        a = stack.pop();
        tmp = 'sin(' + a + ')'
        stack.append(tmp)
    elif token == 'tan':
        a = stack.pop();
        tmp = 'tan(' + a + ')'
        stack.append(tmp)
    elif token == 'pi':
        a = stack.pop();
        b = stack.pop();
        tmp = 'pi'
        stack.append(tmp)
    else:
        try:
            stack.append(token)
        except ValueError:
            raise ValueError(f"{token!r} - неизвестная операция")
return stack.pop()

```

```

def func(rpn_string: str, valueX: int):
    stack = []
    func = ''
    for token in rpn_string.split():
        if token == 'x':
            stack.append(valueX)
        elif token == '+':

```

```

        a = stack.pop();
        b = stack.pop();
        tmp = b + a
        stack.append(tmp)
    elif token == '-':
        a = stack.pop();
        b = stack.pop();
        tmp = b - a
        stack.append(tmp)
    elif token == '/':
        a = stack.pop();
        b = stack.pop();
        tmp = b / a
        stack.append(tmp)
    elif token == '*':
        a = stack.pop();
        b = stack.pop();
        tmp = b * a
        stack.append(tmp)
    elif token == '^':
        a = stack.pop();
        b = stack.pop();
        tmp = b ** a
        stack.append(tmp)
    elif token == 'exp':
        a = stack.pop();
        tmp = m.exp(a)
        stack.append(tmp)
    elif token == 'log':
        a = stack.pop();
        tmp = m.log(a)
        stack.append(tmp)
    elif token == 'sqrt':
        a = stack.pop();
        tmp = m.sqrt(a)
        stack.append(tmp)
    elif token == 'cos':
        a = stack.pop();
        tmp = m.cos(a)
        stack.append(tmp)
    elif token == 'sin':
        a = stack.pop();
        tmp = m.sin(a)
        stack.append(tmp)
    elif token == 'tan':
        a = stack.pop();
        tmp = m.tan(a)
        stack.append(tmp)
    elif token == 'pi':
        a = stack.pop();

```

```

        b = stack.pop();
        tmp = m.pi
        stack.append(tmp)
    else:
        try:
            stack.append(int(token))
        except ValueError:
            raise ValueError(f"{token!r} - неизвестная операция")
    return stack.pop()

def f(x: int):
    return (3 + 2 * (1 - x) ** 2)

def work(f, a, b, n, tmp: str, out):
    print("\nЧисло разбиений: ", n)
    out.write("Число разбиений: " + str(n))
    h = (b-a)/float(n)
    print("Шаг:", h)
    out.write("\nШаг:" + str(h))
    total = sum([f(tmp, a + (k*h)) for k in range(0, n)])
    result = h * total
    print("Результат: ", result)
    out.write("\nРезультат: " + str(result))
    return result

file = open('rpn.txt')
tmp = file.read()
res = toInfix(tmp)
print(res)
file.close()
with open("param.txt") as file:
    for line in file:
        a, b, n = line.split()
out = open("res.txt", "w")
work(func, int(a), int(b), int(n), tmp, out)
out.close()

toRPN.py
def infix_to_postfix(expression):
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3, 'sin': 4, 'cos': 4,
'log': 4, 'tg': 4, 'exp': 4}
    stack = []
    postfix = []
    flag = False
    i = 0
    while i < len(expression):
        char = expression[i]
        if char.isdigit():

```

```

arr = ""
flag = True
while flag:
    if i < (len(expression) - 1):
        tmp = expression[i + 1]
        if not tmp.isdigit():
            flag = False
            char = expression[i]
            arr += char
            i += 1
            break
        char = expression[i]
        arr += char
        i += 1
    postfix.append(arr + ' ')
elif char.isalpha():
    arr = ""
    flag = True
    if char == 'x':
        postfix.append(char + ' ')
        i += 1
    else:
        while flag:
            if i < (len(expression) - 1):
                tmp = expression[i + 1]
                if not tmp.isalpha():
                    flag = False
                    char = expression[i]
                    arr += char
                    i += 1
                    break
            char = expression[i]
            arr += char
            i += 1
        stack.append(arr)
elif char == '(':
    stack.append(char)
    i += 1
elif char == ')':
    while stack and stack[-1] != '(':
        postfix.append(stack.pop() + ' ')
    stack.pop()
    i += 1
else:
    while stack and precedence.get(stack[-1], 0) >= precedence.get(char,
0):
        postfix.append(stack.pop() + ' ')
        stack.append(char)
        i += 1
while stack:

```

```
        postfix.append(stack.pop() + ' ')
    return ''.join(postfix)

file = open("infix.txt")
infix_expression = file.read()
file.close()
postfix_expression = infix_to_postfix(infix_expression)
print(postfix_expression)
out = open("rpn.txt", "w")
out.write(postfix_expression)
out.close()
```