МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБО6РОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподаватель		С.А. Рогачев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

ФС Поста

по курсу: Теория Вычислительных Процессов

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4134к		Столяров Н.С.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Цель работы

Построить формальную систему Поста FSp, реализующую вычисление заданной арифметической функции. Написать программу на языке высокого уровня имитирующую (эмулирующую) вычисления на основе выводимости в формальной системе Поста. W Программа должна работать на любых входных данных из заданного множества. W Программа должна удовлетворять предъявляемым требованиям.

Постановка задачи;

Реализовать функцию x-y+1.

Листинг программы на языке высокого уровня с комментариями;

```
import re

# Фиксированные входные данные
A = ['1', '+', '-', '=']
X = ['y', 'x']
A1 = ['1']
R = [
    "(1) x-y+1= -> x-=y+1",
    "(2) =1x-1 -> =x",
    "(2) x-=y+1 -> x-=y+1",
    "(2) x-=y -> =x-y"
]

rule_patterns = [
    (r'(\d+)-(\d+)\+(\d+)=', r'\1-=\2+\3'),
    (r'=1(\d+)-1', r'=\1'),
    (r'(\d+)-=(\d+)\+(\d+)', r'\1-=\2\3'),
    (r'(\d+)-=(\d+)', r'=\1-\2')
]

# Функция для проверки строки на наличие символов, не входящих в алфавит
def check_alphabet(input_str, alphabet):
    for char in input_str:
        if char not in alphabet:
            raise ValueError(f"Ошибка: Найден символ '{char}', который не входит в алфавит.")

# Функция для проверки строк на наличие переменных, не входящих в заданное множество
def check_variables(input_str, variables):
    found_vars = re.findall(r'[a-zA-Z]+\d*', input_str)  # Найти все переменные
    for var in found_vars:
        if var not in variables:
            raise ValueError(f"Найдена переменная, не входящая в заданное множество: {var}")

# Функция для проверки формата аксиомы
def check_axiom_format(axiom):
    # Проверяем, соответствует ли аксиома формату число*число-число=
    pattern = r'^\d+\-\d++\d+=$'
    if not re.match(pattern, axiom):
```

```
        raise ValueError("Ошибка: Аксиома должна иметь формат число-число+число=.")

# Функция для проверки правил на наличие недопустимых переменных
def check_rules_variables(rules, variables):
    for rule in rules:
        # Извлечение правых частей правил
        parts = rule.split('->')
        for part in parts:
            check_variables(part.strip(), variables)  # Проверка на наличие недопустимых
переменных

# Функция для применения правил с использованием регулярных выражений
def apply_rules(start_str, rules):
    current_str = start_str
    result_log = []

    while True:
        applied = False

        for i, (pattern, replacement) in enumerate(rule_patterns):
            match = re.search(pattern, current_str)
            print(pattern, current_str, match)
            if match:
                result_log.append(f"Исходная строка: {current_str}")
                result_log.append(f"Применяемое правило: {R[i]}")
                current_str = re.sub(pattern, replacement, current_str, count=1)
                result_log.append(f"Результат применения: {current_str}\n")
                applied = True
                print(f"Применено правило: {R[i]}")
                break

        if not applied:
            print(f"Не удалось применить ни одно правило к строке: {current_str}")
            break

    return current_str, result_log

# Основная функция программы
def main(output_file):
    try:
        # Ввод примера через консоль
        example = input("Введите пример в формате число-число+число=: ").strip()

        # Открываем файл для записи и записываем ввод
        with open(output_file, 'w', encoding='utf-8') as f:
            f.write(f"Введённый пример: {example}\n")
            f.write(f"\nВходные данные:\nАлфавит: {A}\nПеременные: {X}\nАксиомы: {A1}\
nПравила: {R}\n\n")

        # Проверка примера на соответствие алфавиту и множеству переменных
        check_alphabet(example, A)
        check_variables(example, X)
```

```
        # check_axiom_format(example)  # Проверка формата аксиомы

        # Применение правил вывода к аксиоме
        current_str, log = apply_rules(example, R)

        # Запись результата в файл
        with open(output_file, 'a', encoding='utf-8') as f:
            f.write("Результаты применения правил:\n")
            for line in log:
                f.write(line + '\n')

            if not log:
                f.write(f"Ни одно правило не было применено для примера: {example}\n")

        print("Вычисление прошло успешно.")

    except ValueError as ve:
        with open(output_file, 'w', encoding='utf-8') as f:
            f.write(f"В ходе вычисления произошла ошибка: {ve}\n")
        print(f"В ходе вычисления произошла ошибка: {ve}")
    except Exception as e:
        with open(output_file, 'w', encoding='utf-8') as f:
            f.write(f"Ошибка: {e}\n")
        print(f"Ошибка: {e}")

# использование программы
if __name__ == '__main__':
    output_file = 'output.txt'  # Файл для сохранения результата
    main(output_file)
```

Содержимое входного файла согласно заданию;

0000000000000000000000000000000000000

Примеры результатов выполнения:

```
Введённый пример: 1111111-1111+1=

Входные данные:
Алфавит: ['1', '+', '-', '=']
Переменные: ['y', 'x']
Аксиомы: ['1']
Правила: ['(1) x-y+1= -> x-=y+1', '(2) =1x-1 -> =x', '(2) x-=y+1 -> x-=y+1', '(2) x-=y -> =x-y']

Результаты применения правил:
Исходная строка: 1111111-1111+1=
Применяемое правило: (1) x-y+1= -> x-=y+1
Результат применения: 1111111-=1111+1
```

Исходная строка: 1111111-=1111+1
Применяемое правило: (2) x-=y+1 -> x-=y+1
Результат применения: 1111111-=11111

Исходная строка: 1111111-=11111
Применяемое правило: (2) x-=y -> =x-y
Результат применения: =1111111-11111

Исходная строка: =1111111-11111
Применяемое правило: (2) =1x-1 -> =x
Результат применения: =1111111111

Вывод

Была построена программа по формальной системе Поста