Федеральное государственное автономное образовательное учреждение

высшего образования «Санкт-Петербургский государственный университет аэрокосмического приборостроения» (ГУАП)

ОТЧЕТ ЗАЩИЩЕН

Ст. преподаватель Рогачев С.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Должность Ф.И.О дата, подпись

**Отчет**

**о лабораторной работе №3**

**«Алгоритмическая система Поста»**

по дисциплине «Теория вычислительных процессов»

ОТЧЕТ ВЫПОЛНИЛ:

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф.И.О. группа дата, подпись

Санкт-Петербург

2024

**Цель работы**

Изучить и построить формальную систему Поста, реализующую вычисление заданной арифметической функции.

**Постановка задачи**

* 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.
* Программа должна работать на любых входных данных из заданного множества.
* Программа должна удовлетворять предъявляемым требованиям.

Вариант:



**Листинг программы на языке высокого уровня с комментариями**

|  |
| --- |
| import re  # Получение индекса переменной в строке  def get\_index(found\_variables, ch):      return found\_variables.find(ch)  # Проверка корректности входных данных  def is\_input\_correct(function, alphabet, variables, axioms, left\_rules, right\_rules):      def check\_symbols(s):          for character in s:              if character not in alphabet:                  print(f"Ошибка: символ '{character}' не входит в алфавит!")                  return False          return True      if not check\_symbols(function) or not check\_symbols(axioms):          return False      for rule in left\_rules:          for character in rule:              if character not in variables and character not in alphabet:                  print(f"Ошибка: символ '{character}' не входит в алфавит или множество переменных!")                  return False      return True  # Чтение и разбор входных данных из файла  def read\_input(filename):      try:          with open(filename, 'r', encoding='utf-8') as fi:              function = fi.readline().strip()              alphabet = fi.readline().strip()              variables = fi.readline().strip()              axioms = fi.readline().strip()              # Очистка данных              def clean\_input(s, prefix, suffix):                  return s[len(prefix):-len(suffix)].replace(',', '')              alphabet = clean\_input(alphabet, "A={", "};")              variables = clean\_input(variables, "X={", "};")              axioms = clean\_input(axioms, "A1={", "};")              # Считывание правил преобразований              left\_rules, right\_rules = [], []              for line in fi:                  line = line.strip()                  if line == "}":                      break                  if "->" in line:                      pos = line.find("->")                      left\_rules.append(line[:pos])                      right\_rules.append(line[pos + 2:])              if left\_rules:                  left\_rules[0] = left\_rules[0][3:]              if right\_rules:                  right\_rules[-1] = right\_rules[-1][:-1]              return function, alphabet, variables, axioms, left\_rules, right\_rules      except FileNotFoundError:          print("Ошибка открытия файла!")          return None  # Функция для применения правил  def apply\_rules(function, variables, left\_rules, right\_rules):      try:          with open("D:/GIT/VuzUC/7SEM/TVP/LAB3/output.txt", "w", encoding="utf-8") as out:              rule\_counter = 0              return\_flag = 0              found\_variables = ""              while return\_flag < len(left\_rules):                  reg\_ex = ""                  found\_variables = ""                  for ch in left\_rules[rule\_counter]:                      if ch in variables:                          if ch not in found\_variables:                              reg\_ex += "(1\*)"                              found\_variables += ch                          else:                              reg\_ex += f"(\\{get\_index(found\_variables, ch) + 1})"                      elif ch in "+=":                          reg\_ex += re.escape(ch)                      else:                          reg\_ex += ch                  match = re.search(reg\_ex, function)                  if match:                      out.write(f"Исходная строка: {function}\n")                      out.write(f"Применяемое правило: {left\_rules[rule\_counter]} -> {right\_rules[rule\_counter]}\n")                      return\_flag = 0                      tmp = ""                      values = [match.group(i) for i in range(1, len(match.groups()) + 1)]                      for ch in right\_rules[rule\_counter]:                          if ch in variables:                              tmp += values[get\_index(found\_variables, ch)]                          else:                              tmp += ch                      function = match.string[:match.start()] + tmp + match.string[match.end():]                      out.write(f"Результат применения правила: {function}\n\n")                  else:                      return\_flag += 1                  rule\_counter = (rule\_counter + 1) % len(left\_rules)              print("Вычисления завершены")      except IOError:          print("Ошибка открытия файла для записи!")  # Главная функция  if \_\_name\_\_ == "\_\_main\_\_":      # Инициализация переменных      data = read\_input("D:/GIT/VuzUC/7SEM/TVP/LAB3/input.txt")      if data:          function, alphabet, variables, axioms, left\_rules, right\_rules = data          if is\_input\_correct(function, alphabet, variables, axioms, left\_rules, right\_rules):              apply\_rules(function, variables, left\_rules, right\_rules)  import re  # Получение индекса переменной в строке  def get\_index(found\_variables, ch):      return found\_variables.find(ch)  # Проверка корректности входных данных  def is\_input\_correct(function, alphabet, variables, axioms, left\_rules, right\_rules):      def check\_symbols(s):          for character in s:              if character not in alphabet:                  print(f"Ошибка: символ '{character}' не входит в алфавит!")                  return False          return True      if not check\_symbols(function) or not check\_symbols(axioms):          return False      for rule in left\_rules:          for character in rule:              if character not in variables and character not in alphabet:                  print(f"Ошибка: символ '{character}' не входит в алфавит или множество переменных!")                  return False      return True  # Чтение и разбор входных данных из файла  def read\_input(filename):      try:          with open(filename, 'r', encoding='utf-8') as fi:              function = fi.readline().strip()              alphabet = fi.readline().strip()              variables = fi.readline().strip()              axioms = fi.readline().strip()              # Очистка данных              def clean\_input(s, prefix, suffix):                  return s[len(prefix):-len(suffix)].replace(',', '')              alphabet = clean\_input(alphabet, "A={", "};")              variables = clean\_input(variables, "X={", "};")              axioms = clean\_input(axioms, "A1={", "};")              # Считывание правил преобразований              left\_rules, right\_rules = [], []              for line in fi:                  line = line.strip()                  if line == "}":                      break                  if "->" in line:                      pos = line.find("->")                      left\_rules.append(line[:pos])                      right\_rules.append(line[pos + 2:])              if left\_rules:                  left\_rules[0] = left\_rules[0][3:]              if right\_rules:                  right\_rules[-1] = right\_rules[-1][:-1]              return function, alphabet, variables, axioms, left\_rules, right\_rules      except FileNotFoundError:          print("Ошибка открытия файла!")          return None  # Функция для применения правил  def apply\_rules(function, variables, left\_rules, right\_rules):      try:          with open("D:/GIT/VuzUC/7SEM/TVP/LAB3/output.txt", "w", encoding="utf-8") as out:              rule\_counter = 0              return\_flag = 0              found\_variables = ""              while return\_flag < len(left\_rules):                  reg\_ex = ""                  found\_variables = ""                  for ch in left\_rules[rule\_counter]:                      if ch in variables:                          if ch not in found\_variables:                              reg\_ex += "(1\*)"                              found\_variables += ch                          else:                              reg\_ex += f"(\\{get\_index(found\_variables, ch) + 1})"                      elif ch in "+=":                          reg\_ex += re.escape(ch)                      else:                          reg\_ex += ch                  match = re.search(reg\_ex, function)                  if match:                      out.write(f"Исходная строка: {function}\n")                      out.write(f"Применяемое правило: {left\_rules[rule\_counter]} -> {right\_rules[rule\_counter]}\n")                      return\_flag = 0                      tmp = ""                      values = [match.group(i) for i in range(1, len(match.groups()) + 1)]                      for ch in right\_rules[rule\_counter]:                          if ch in variables:                              tmp += values[get\_index(found\_variables, ch)]                          else:                              tmp += ch                      function = match.string[:match.start()] + tmp + match.string[match.end():]                      out.write(f"Результат применения правила: {function}\n\n")                  else:                      return\_flag += 1                  rule\_counter = (rule\_counter + 1) % len(left\_rules)              print("Вычисления завершены")      except IOError:          print("Ошибка открытия файла для записи!")  # Главная функция  if \_\_name\_\_ == "\_\_main\_\_":      # Инициализация переменных      data = read\_input("D:/GIT/VuzUC/7SEM/TVP/LAB3/input.txt")      if data:          function, alphabet, variables, axioms, left\_rules, right\_rules = data          if is\_input\_correct(function, alphabet, variables, axioms, left\_rules, right\_rules):              apply\_rules(function, variables, left\_rules, right\_rules) |

**Содержимое входного файла согласно заданию**

111+11=

A={1,+,=};

X={x,y};

A1={1};

R={x+y=->xxy}

input.txt содержит входные данные:

* Функция (начальная строка).
* Алфавит (множество допустимых символов).
* Переменные (для подстановок в правилах).
* Аксиомы (начальные условия).
* Правила преобразований (левых и правых частей).

output.txt записывает историю работы программы:

* Исходная строка.
* Применённое правило.
* Результат после применения правила.

Как программа проверяет корректность входных данных?

1. Проверяется, что все символы в функции, аксиомах и правилах присутствуют в алфавите или множестве переменных.
2. Если встречается символ вне алфавита, программа выводит сообщение об ошибке и завершает выполнение.

Правила преобразования:

левая\_часть -> правая\_часть

1. Программа ищет в функции соответствие левой части правила.
2. При совпадении левая часть заменяется на правую.
3. Регулярные выражения используются для нахождения и подстановки переменных

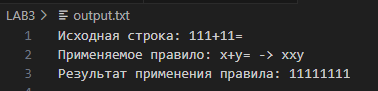
Функция apply\_rules:

1. Строит регулярное выражение для поиска левой части правила в текущей строке.
2. При нахождении совпадения применяет правую часть правила, заменяя найденное выражение.
3. Повторяет процесс до тех пор, пока ни одно правило не применимо.

Переменные заменяются на группы символов из регулярного выражения. Например, если переменная x соответствует 111, то она подставляется в правую часть правила как 111.

**Примеры результатов выполнения**

**Примеры c корректными входными данными**



**Примеры c некорректными входными данными**

|  |
| --- |
| **111+a=**  **A={1,+,=};**  **X={x,y};**  **A1={1};**  **R={a+y=->xxy}** |





**Вывод**

Изучена и построена формальная система Поста, реализующая вычисление заданной арифметической функции.