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 №2**

**«Изучение принципов функционирования машины Тьюринга»**

по дисциплине «Теория вычислительных процессов»

ОТЧЕТ ВЫПОЛНИЛ:

Студент Самарин Д. В. 4134К \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф.И.О. группа дата, подпись

Санкт-Петербург

2024

**Цель работы:** изучить принципы функционирования машины Тьюринга.

**Основные сведения из теории:**

Машина Тьюринга – это абстрактный автомат, предназначенный для выполнения вычислений алгоритмов. Она состоит из трех основных компонентов:

1. **Управляющее устройство (УУ)** — может находиться в одном из множества внутренних состояний, представляющих внутренний алфавит Q={q0,q1,...,qn,qz}
2. **Бесконечная лента** — разбита на ячейки, каждая из которых может содержать символ из внешнего алфавита A={a1,a2,...,am,λ}, где λ — пустой символ.
3. **Головка** — устройство для чтения и записи, которое передвигается по ленте, считывает или записывает символы в текущую ячейку ленты.

Каждое дискретное действие машины Тьюринга выполняется в соответствии с командой, определяющей:

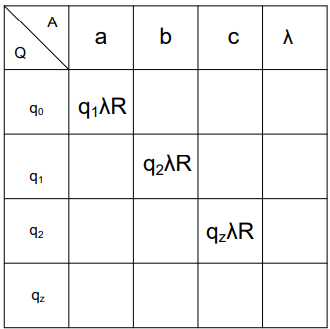
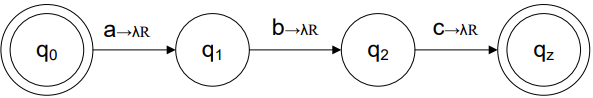
* текущее состояние;
* символ в текущей ячейке;
* новый символ для записи;
* новое состояние;
* направление движения головки (вправо, влево, или на месте).

Команды записываются в виде переходов qiaj→qkaldq, где qi​ и qk​ — состояния до и после выполнения команды, aj​ и al​ — символы на ленте, а d — направление сдвига.

Полное состояние машины называется конфигурацией и включает текущее состояние и символы на ленте. Стандартные конфигурации:

* **Начальная** — головка смотрит на левый край символов, и машина в начальном состоянии q0​.
* **Конечная** — машина в конечном состоянии qz​, и дальнейшее выполнение команд невозможно.

Алгоритмы для машины Тьюринга можно задать тремя способами:

1. Перечислением команд
2. Таблицей переходов
3. Блок-схемой

Функция называется вычислимой по Тьюрингу, если существует машина Тьюринга, правильно вычисляющая значения функции для любого входа. В случае зацикливания функция считается неопределенной.

Тезис Тьюринга утверждает, что любая вычислимая функция может быть реализована на машине Тьюринга, и если для функции нельзя создать алгоритм, то она не вычислима в принципе.

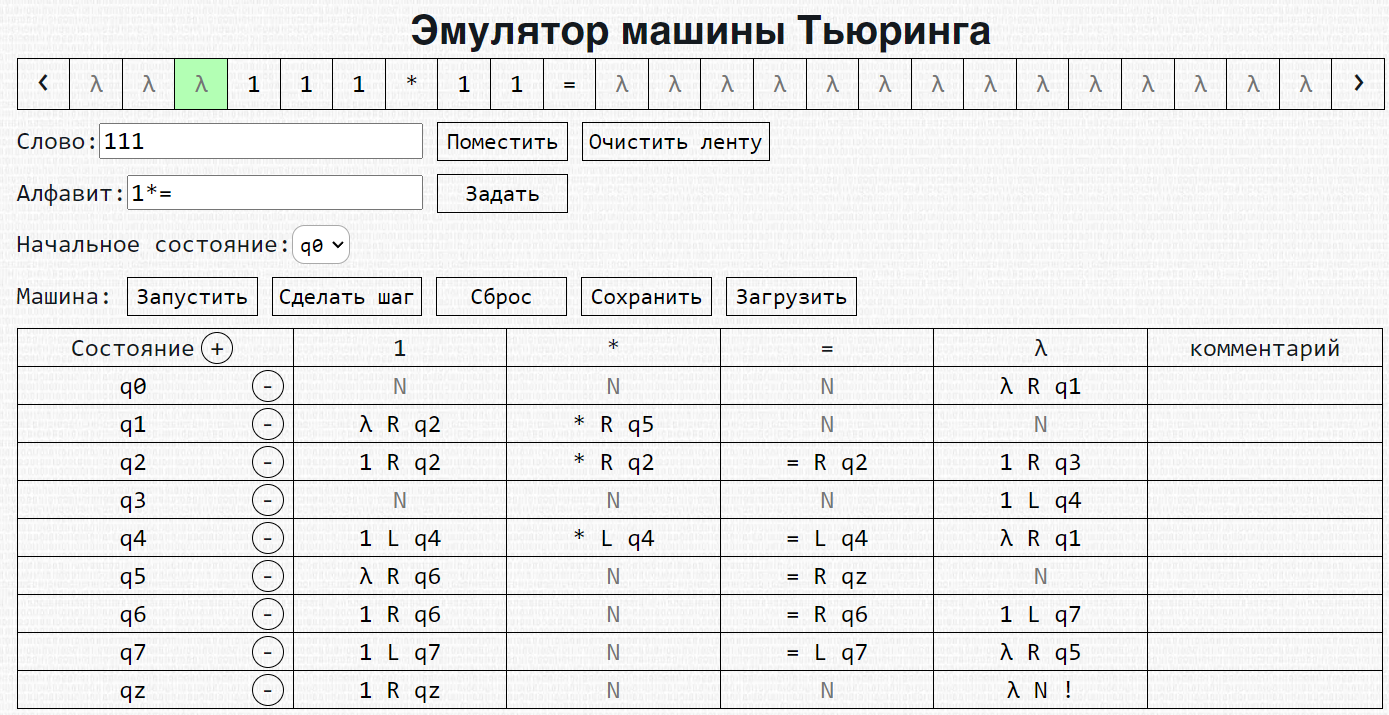
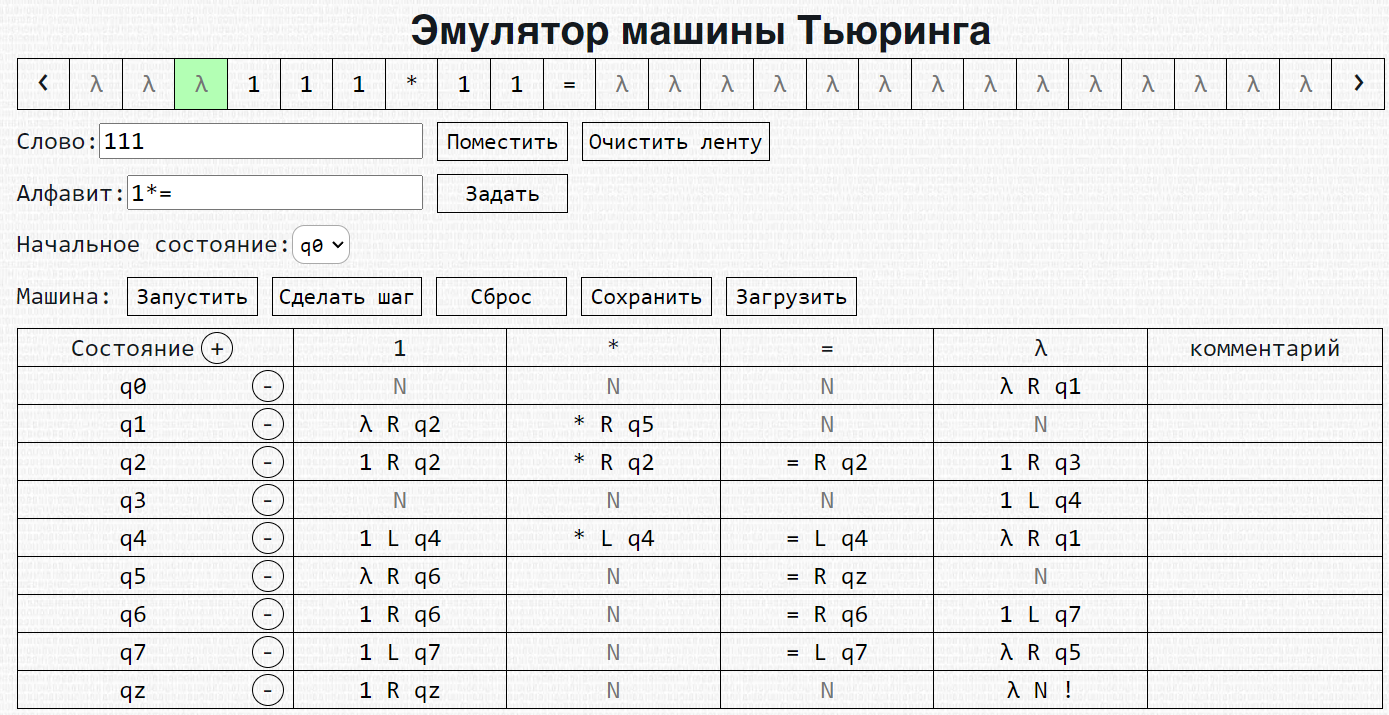
**Постановка задачи**

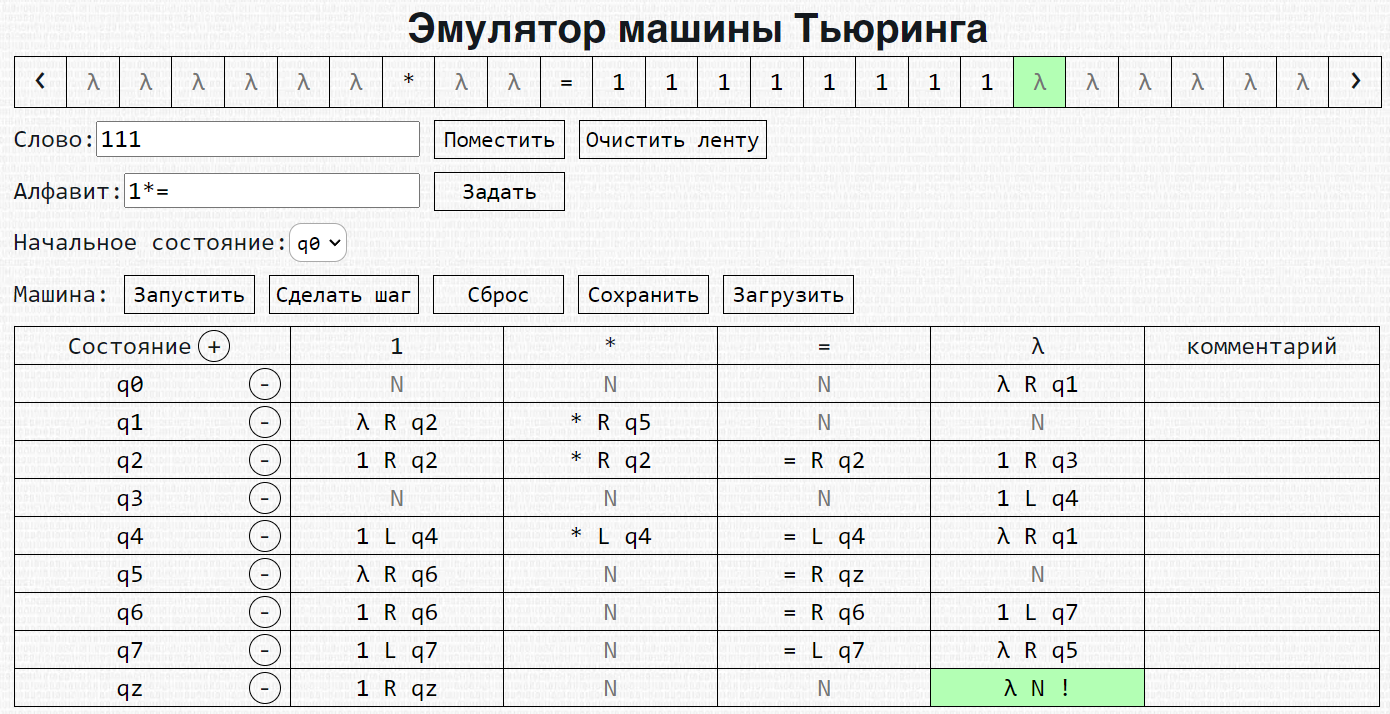
Необходимо написать программу для машины Тьюринга, реализующую вычисление арифметической функции согласно выданному варианту задания. Должна быть составлена совокупность команд P. Для выполнения данного задания следует использовать приложение Algo2000.

**Вариант:**



**Совокупность команд для машины Тьюринга**

****

****

q0,\_ -> \_,R,q1

q1,1 -> \_,R,q2

q1,\* -> \*,R,q5

q2,1 -> 1,R,q2

q2,\* -> \*,R,q2

q2,= -> =,R,q2

q2,\_ -> 1,R,q3

q3,\_ -> 1,L,q4

q4,1 -> 1,L,q4

q4,\* -> \*,L,q4

q4,= -> =,L,q4

q4.\_ -> \_,R,q1

q5,1 -> \_,R,q6

q5,= -> =,R,qz

q6,1 -> 1,R,q6

q6,= -> =,R,q6

q6,\_ -> 1,L,q7

q7,1 -> 1,L,q7

q7,= -> =,L,q7

q7,\_ -> \_,R,q5

qz,1 -> 1,R,qz

qz,\_ -> \_,N,!

Исходное состояние ленты tape.txt:

\_111\*11=\_

* **111**: соответствует значению x1=3 (три единицы).
* **\***: маркер, который разделяет области на ленте.
* **11**: соответствует значению x2=2 (две единицы).
* **=**: служит разделителем, чтобы машина Тьюринга могла различать части данных.
* **\_**: пустые символы для обозначения границ ленты.

Таким образом, на ленте: x1=3, x2=2.

Программа находит и обрабатывает данные на ленте в несколько этапов:

Этап 1: Удаление символов 1 слева от \*:

q1,1 -> \_,R,q2

Этап 2: Перемещение к разделителю = и пустой ячейке:

q2,1 -> 1,R,q2

q2,\* -> \*,R,q2

q2,= -> =,R,q2

q2,\_ -> 1,R,q3

Этап 3: Добавление символов 1 (удваивание x1​)

q3,\_ -> 1,L,q4

Машина записывает символ 1 в пустые ячейки, чтобы **удвоить x1​**.

Возврат и обработка x\_2:

q5,1 -> \_,R,q6

q6,1 -> 1,R,q6

q6,\_ -> 1,L,q7

Машина добавляет символы 1 из второй группы (значение x2​) к уже удвоенному x1​.

Машина завершает выполнение в терминальном состоянии qz после обработки всех символов. На ленте остаётся результат вычисления.

Итог:

2 \* 3 + 2 = 8

\_\_\_\_\*\_\_=11111111\_

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

|  |
| --- |
| import sys  # импортируем модуль sys для работы с системными сообщениями и завершением программы  # класс для реализации машины Тьюринга  class TuringMachine:      def \_\_init\_\_(self, tape, program, alphabet):          # инициализация машины Тьюринга          self.tape = list(tape)  # лента представлена в виде списка символов          self.head\_position = 0  # начальная позиция головки на ленте          self.state = 'q0'  # начальное состояние          self.program = program  # программа с командами для машины Тьюринга          self.alphabet = alphabet  # алфавит допустимых символов          self.history = []  # история выполнения команд для вывода в файл      # метод для перемещения головки машины      def move\_head(self, direction):          if direction == 'R':  # если указано движение вправо              self.head\_position += 1              # если головка выходит за правый край ленты, расширяем её              if self.head\_position >= len(self.tape):                  self.tape.append('\_')  # добавляем пустой символ '\_'          elif direction == 'L':  # если указано движение влево              self.head\_position -= 1              # если головка выходит за левый край, расширяем ленту слева              if self.head\_position < 0:                  self.tape.insert(0, '\_')                  self.head\_position = 0  # сдвигаем позицию головки на начало      # метод выполнения программы машины Тьюринга      def execute(self):          # пока текущее состояние не является терминальным ('!')          while self.state != '!':              # получаем символ, на котором находится головка              current\_symbol = self.tape[self.head\_position]              # ищем команду для текущего состояния и символа              command = self.program.get((self.state, current\_symbol))              # если команда не найдена, записываем ошибку и выходим из цикла              if not command:                  self.history.append(f"Ошибка: нет перехода для ({self.state}, {current\_symbol})")                  break              # распаковываем команду: новый символ, направление движения, новое состояние              new\_symbol, direction, new\_state = command              # записываем текущее состояние ленты и выполняемую команду для истории              tape\_before = ''.join(self.tape)  # состояние ленты перед выполнением команды              head\_pos = ' ' \* self.head\_position + '^'  # позиция головки на ленте              command\_str = f"{self.state},{current\_symbol} -> {new\_symbol},{direction},{new\_state}"              # добавляем запись в историю              self.history.append(f"{tape\_before}\n{head\_pos}\n{command\_str}")              # выполняем команду: заменяем символ на новый и двигаем головку              self.tape[self.head\_position] = new\_symbol              self.move\_head(direction)              self.state = new\_state  # переходим в новое состояние      # метод для сохранения результата работы в файл      def save\_output(self, filename):          with open(filename, 'w') as f:              # записываем историю выполнения команд              for line in self.history:                  f.write(line + '\n')              # записываем итоговое состояние ленты              f.write("Итоговое состояние ленты: " + ''.join(self.tape) + '\n')  # функция для загрузки содержимого файла  def load\_file(filename):      with open(filename, 'r') as f:          return f.read().strip()  # считываем содержимое и удаляем лишние пробелы  # функция для загрузки программы из файла  def load\_program(filename):      program = {}      with open(filename, 'r') as f:          for line in f:              line = line.strip()  # убираем лишние пробелы по краям              if not line or '->' not in line:                  continue  # пропускаем пустые строки и строки без '->'              # разделяем строку на левую и правую части по разделителю '->'              left, right = line.split('->')              left = left.strip()              right = right.strip()              try:                  # разбиваем левую часть на состояние и символ                  state, symbol = left.split(',')                  # разбиваем правую часть на новый символ, направление и новое состояние                  new\_symbol, direction, new\_state = right.split(',')                  # сохраняем команду в словарь                  program[(state.strip(), symbol.strip())] = (new\_symbol.strip(), direction.strip(), new\_state.strip())              except ValueError:                  print(f"Ошибка в строке программы: '{line}'")                  sys.exit(1)  # завершаем выполнение, если ошибка при разборе строки      return program  # функция для загрузки алфавита из файла  def load\_alphabet(filename):      with open(filename, 'r') as f:          return set(f.read().strip().split(','))  # считываем и создаём множество символов  # загрузка данных из файлов  tape = load\_file('LAB2/tape.txt')  # лента  program = load\_program('LAB2/program.txt')  # программа  alphabet = load\_alphabet('LAB2/alphabet.txt')  # алфавит  # проверка символов на ленте на принадлежность алфавиту  for symbol in tape:      if symbol not in alphabet:          sys.exit(f"Ошибка: символ '{symbol}' не принадлежит алфавиту")  # создание машины Тьюринга и выполнение программы  tm = TuringMachine(tape, program, alphabet)  tm.execute()  tm.save\_output('output.txt')  # сохранение результата в файл  print("Выполнение завершено. Результаты сохранены в output.txt.") |

**Пример результата выполнения**

****

**Вывод:** в ходе лабораторной работы были получены навыки создания алгоритмов на машине Тьюринга.