Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ульяновский государственный технический университет»
Кафедра «Вычислительная техника»

Теория автоматов

Лабораторная работа №4
«Элементы языка сценариев»

Выполнил
Студент группы ИВТбд-21
Ведин В. А.
Проверил(а):
ст. преподаватель кафедры «ВТ»
Лылова А.В.

Ульяновск
2024

ЗАДАНИЕ

Требуется выполнить программную реализацию автомата типа мили, выполняющего программы, написанные на языке сценариев. Язык сценариев должен содержать конструкции для объявления переменных, присваивания им значений или результатов вычисления выражений, условный оператор, а также инструменты ввода с клавиатуры и вывода на экран значений переменных. Выражение может содержать как заданные числовые значения, так и переменные. Условный оператор должен выполнять тот или иной фрагмент кода в зависимости от логического выражения. Логическое выражение должно содержать один знак условия (равно, не равно, больше, меньше, больше или равно, меньше или равно), слева и справа от которого стоят числовые значения или арифметические выражения. При обнаружении ошибки в языковых конструкциях или в арифметических выражениях пользователю должно выводится сообщение, содержащее номер строки с ошибкой. Для вычисления арифметических выражения используется класс, реализованный в лабораторной работе №3 в соответствии с вариантом задания: операции в целых числа в диапазоне от -128 до 127.

ТЕКСТОВОЕ ОПИСАНИЕ РЕАЛИЗОВАННОГО АВТОМАТА

На вход автомату подается файл code.txt, содержащий в себе любое количество строк. Автомат построчно считывает файл, проверяя какая команда встречается ему, также проверяет арифметическую корректность, после чего считает арифметическое выражение, если оно верно, также рассчитывает результат логических условий. В случае неудачи выводит ошибку.

При выполнении автомата, программа выводит свои состояния в файл out_state.txt, а в результате работы в консоль выводится либо результат, либо ошибка.

ОПИСАНИЕ КОНСТРУКЦИЙ ЯЗЫКА

Таблица 1. Описание конструкции языка сценариев

Действие	Синтаксис
Присвоение значения	<variable_name> = <num (int)> || <str> || <arithmetic expression>
Условная конструкция if	if (<condtion>) { <body> }
Условная конструкция if else	if (<condition>) { <body> } else { <body> }
Ввод	<variable_name> = getdata()
Вывод	outdata(<variable_name> || <num (int)> || <str> || <arithmetic expression>
Арифметические операции	+, -, *, /, =
Логические операции	==, <, <=, >, >=, !=

ТАБЛИЦЫ ОБОЗНАЧЕНИЙ

Таблица 2. Обозначение входных сигналов.

Входной сигнал	Обозначение
x1	for line in code:
x2	if line.startwith(“if “):
x3	try:
x4	elif “} else {“ in line:
x5	elif “}” in line:
x6	elif len(line) == 0 or len(bool_expressions_results) > 0 and not bool_expressions_results[-1]:
x7	elif “ = “ in line:
x8	if len(line) != 2:
x9	if not line[0][0].isalpha() and “getdata” not in line[1] and “”” not in line[1]:
x10	If “getdata” in line[1]:
x11	If “”” in line[1]:
x12	if e != -1:
x13	elif line.startwith(“outdata”):

Таблица 3. Обозначение выходных фукнций.

Выходная функция	Обозначение
y1	code = open(“code.txt”, “r”)
y2	bool_expressions_results = []
y3	num_line += 1

Выходная функция	Обозначение
y4	line = line.strip().rstrip()
y5	expression = get_variable(line)
y6	result = calculate_bool_expression(expression)
y7	bool_expressions_results.append(int(result))
y8	print_error(e)
y9	bool_expressions_results.append(1 — bool_expressions_results.pop())
y10	bool_expressions_results.pop()
y11	line = line.split(“ = ”)
y12	print_error(“Invalid variable definition”)
y13	print_error(“Variable name must start with a letter”)
y14	variables[line[0]] = input()
y15	variables[line[0]] = line[1].replace(‘””, ‘“”)
y16	e = -1
y17	result = Calc(line[1]).calculate()
y18	e = er
y19	result = calculate_bool_expression(line[1])
y20	variables[line[0]] = result
y21	print(get_value(line))
y22	print_error(“Invalid command”)

БЛОК-СХЕМА

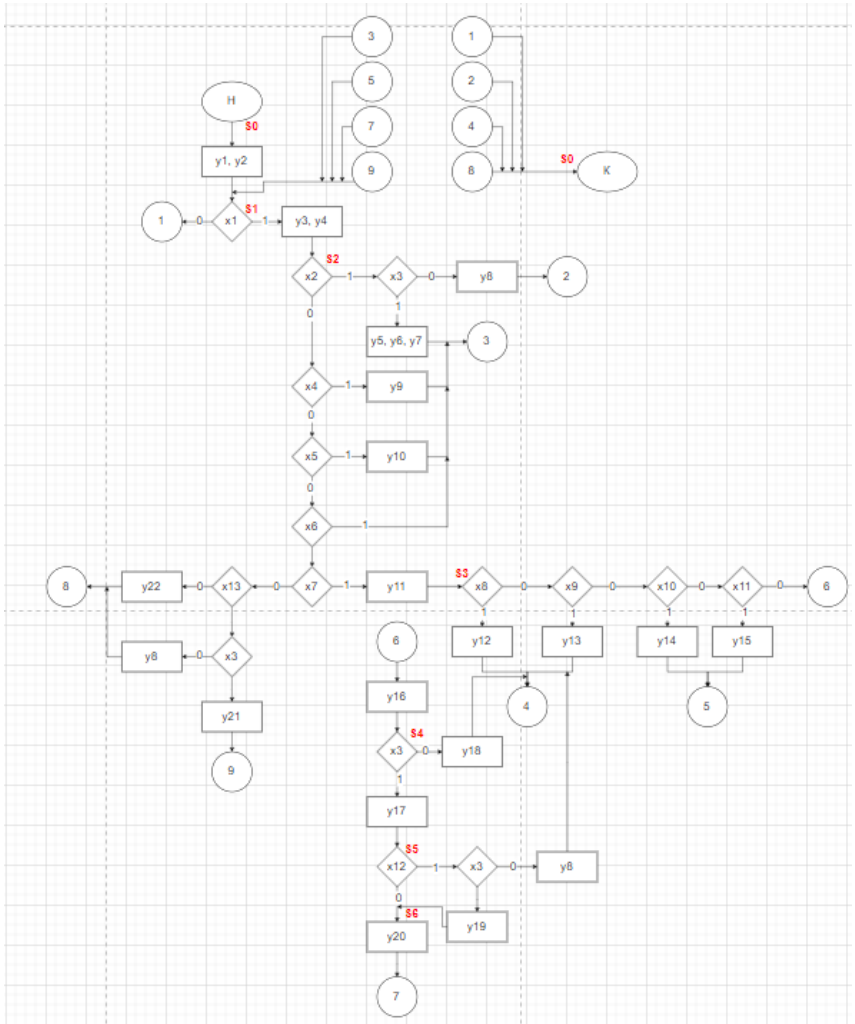

Рис 1.1. Блок-схема автомата типа Мили.

ТАБЛИЦА ПЕРЕХОДОВ И ВЫХОДОВ АВТОМАТА

Таблица 4. Переходы и выходы автомата

Начальное состояние	Конечное состояние	Входные функции	Выходные функции
S0	S1	1	y1, y2
S1	S0	!x1	-
S1	S2	x1	y3, y4
S2	S0	x2x3	y8
S2	S1	x2x3	y5, y6, y7
S2	S1	!x2x4	y9
S2	S1	!x2!x4x5	y10
S2	S1	!x2!x4!x5x6	-
S2	S3	!x2!x4!x5!x6x7	y11
S2	S1	!x2!x4!x5!x7x13x3	y21
S2	S0	!x2!x4!x5!x6!x7!x13	y22
S2	S0	!x2!x4!x5!x6!x7x13!x3	y8
S3	S0	x8	y12
S3	S0	!x8x9	y13
S3	S1	!x8!x9x10	y14
S3	S1	!x8!x9!x10x11	y15
S3	S4	!x8!x9!x10!x11	y16
S4	S0	!x3	y18

S4	S5	x3	y17
Начальное состояние	Конечное состояние	Входные функции	Выходные функции
S5	S0	x12!x3	y8
S5	S6	!x12	-
S5	S6	x12x3	y19
S6	S1	1	y20

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

```
a = 12
b = 120
c = 1200
d = "hello"
e = getdata()

outdata(e)
outdata(d)

if (b > a) {
    outdata("b is more than a")
    if (c > b) {
        outdata("c is more than b")
    }
    outdata("Hi im good!")
} else {
    outdata("a is more than b")
}
outdata("Goodbye!")
```

Рис 2.1. Тест 1 – без ошибок. Code.txt.

```
Hello world!
Hello world!
hello
b is more than a
c is more than b
Hi im good!
Goodbye!
```

Рис 2.2. Тест 1 – без ошибок. Вывод

n.py code.txt out_states.txt ×

```
S0 -> S1 -> S2 -> S3 -> S4 -> S5 -> S6 -> S1 -> S2 -> S3 -> S4 -> S5 -> S6 -> S1 -> S2 -> S3 -> S4 -> S5 -> S6 -> S1 ->
S2 -> S3 -> S2 -> S3 -> S2 -> S2 -> S1 -> S2 -> S1 -> S2 -> S2 -> S1 -> S2 -> S1 -> S2 -> S1 -> S2 -> S1 -> S2 -> S2 ->
S1 -> S2 -> S2 -> S2 -> S2 -> S1 -> S0
```

Рис 2.3. Тест 1 – без ошибок. Состояния

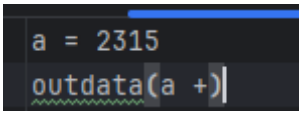

Рис 3.1. Тест 2 – ошибка в выражении. Code.txt

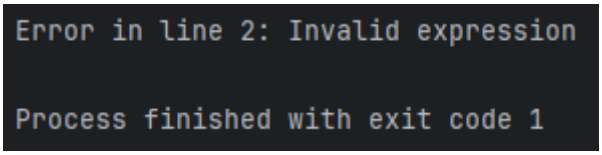

Рис 3.2. Тест 2 – ошибка в выражении. Вывод

Рис 3.3. Тест 2 – ошибка в выражении. Состояния

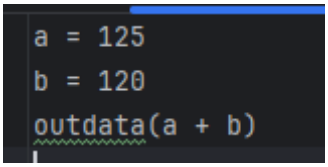

Рис 4.1. Тест 3 – переполнение. Code.txt

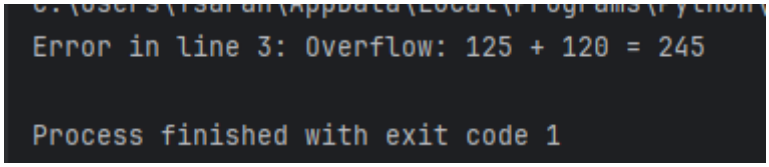

Рис 4.2. Тест 3 – переполнение. Вывод

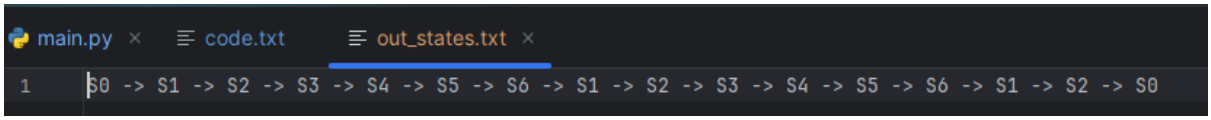

Рис 4.3. Тест 3 – переполнение. Состояния

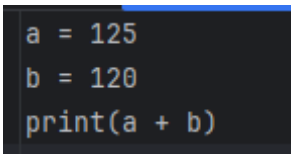

Рис 5.1. Тест 4 – неизвестная команда. Code.txt

```
Error in line 3: Invalid command

Process finished with exit code 1
```

Рис 5.2. Тест 4 – неизвестная команда. Вывод

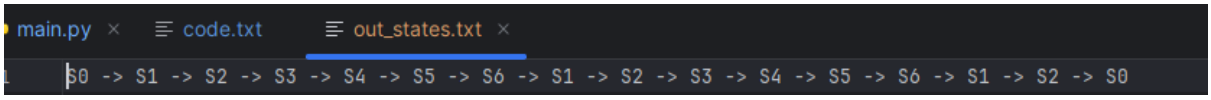

Рис 5.3. Тест 4 – неизвестная команда. Состояния

ИТОГИ

По итогу данной работы был спроектирован собственный язык сценариев, а также и реализован автомат типа Мили, выполняющий программы, написанные на данном языке. Все арифметические подсчёты данный автомат выполнял с помощью другого автомата, который был реализован в лабораторной работе №3.

ПРИЛОЖЕНИЕ

```
from typing import Any
num_line = 0
variables = dict()
output = open("out_states.txt", "w", encoding="utf-8")
def print_state(state):
    print(f"S{state}{'' if state == 0 else ' -> '}", file=output, end='')
def isdigit(num: str) -> bool:
    try:
        int(num)
        return True
    except ValueError:
        return False
def replace_variable_to_value(string) -> str:
    new_string = []
    for i in range(len(string)):
        if string[i] in variables:
            new_string.append(str(variables[string[i]]))
        elif not isdigit(string[i]) and string[i] not in \
                {"-", "+", "*", "/", "(", ")", ".", " ", ">", "<", "==",
"<=", ">=", "&", "|", "!"}:
            raise ValueError(f"Variable <{string[i]}> is not defined")
        else:
            new_string.append(string[i])
    return ''.join(new_string)
class Calc:
    __math_expression = ""
    __reverse_polish_notation = []
    def __init__(self, math_expression) -> None:
        self.__math_expression = math_expression
        self.__math_expression =
replace_variable_to_value(self.__math_expression)
    def __check_expression(self) -> bool:
        if (not all(char.isdigit() or char in "+-*/" or char.isspace() or
char in "()"
                    for char in self.__math_expression)):
            return False
        stack = []
        reverse_polish_notation = []
        prev_char = ""
        for char in self.__math_expression:
            if char == "(":
                if prev_char != " " and prev_char != "":
                    return False
                stack.append(char)
            elif char == ")":
                while stack and stack[-1] != "(":
                    reverse_polish_notation.append(stack.pop())
                if not stack:
                    return False
                stack.pop()
            elif char in "+-*/":
                if prev_char != ' ' and prev_char != "(" and (prev_char !=
"" and char == "-"):
                    return False
                stack.append(char)
```

```
            elif char.isdigit():
                if prev_char.isdigit():
                    reverse_polish_notation[-1] += char
                    prev_char = char
                    continue
                if prev_char == '-':
                    reverse_polish_notation.append(stack.pop() + char)
                    prev_char = char
                    continue
                if prev_char != " " and prev_char != "" and prev_char !=
"(":
                    return False
                reverse_polish_notation.append(char)
            prev_char = char
        if any(char == "(" for char in stack):
            return False
        while stack:
            reverse_polish_notation.append(stack.pop())
        self.__reverse_polish_notation = reverse_polish_notation
        return True
    def calculate(self) -> int:
        if not self.__check_expression():
            raise ValueError("Invalid expression")
        reverse_polish_notation = self.__reverse_polish_notation
        stack = []
        for item in reverse_polish_notation:
            if isdigit(item):
                stack.append(int(item))
            else:
                if len(stack) < 2:
                    raise ValueError("Invalid expression")
                num2 = stack.pop()
                num1 = stack.pop()
                if num1 > 127 or num1 < -128:
                    raise ValueError("The operand is outside the range: " +
str(num1))
                if num2 > 127 or num2 < -128:
                    raise ValueError("The operand is outside the range: " +
str(num2))
                if item == "+":
                    if num1 + num2 > 127 or num1 + num2 < -128:
                        raise ValueError(f"Overflow: {num1} + {num2} =
{num1 + num2}")
                    stack.append(num1 + num2)
                elif item == "-":
                    if num1 - num2 > 127 or num1 - num2 < -128:
                        raise ValueError(f"Overflow: {num1} - {num2} =
{num1 - num2}")
                    stack.append(num1 - num2)
                elif item == "*":
                    if num1 * num2 > 127 or num1 * num2 < -128:
                        raise ValueError(f"Overflow: {num1} * {num2} =
{num1 * num2}")
                    stack.append(num1 * num2)
                elif item == "/":
                    if num2 == 0:
                        raise ValueError("Division by zero")
```

```
                    stack.append(num1 // num2)
        return stack[-1]
def get_value(line) -> str | int:
    variable_name = get_variable(line)
    if '"' in variable_name:
        return variable_name.replace('"', "")
    if variable_name in variables:
        return variables[get_variable(line)]
    try:
        temp = Calc(variable_name)
        res = temp.calculate()
    except Exception as e:
        raise ValueError(e)
    return res
def print_error(e) -> None:
    print(f"Error in line {num_line}: {e}")
    print_state(0)
    exit(1)
def get_variable(line) -> str:
    return line[line.index("(") + 1:line.rindex(")")]
def calculate_bool_expression(expression) -> str:
    expression = expression.replace("(", " ( ").replace(")", " ) ").split()
    expression = replace_variable_to_value(expression)
    if len({">", "<", "==", ">=", "<=", "!="} & set(expression)) == 0:
        raise ValueError("Invalid boolean expression")
    expression = [str(item) for item in expression]
    expression = eval(" ".join(expression))
    return expression
def main() -> None:
    print_state("0")
    global num_line
    code = open("code.txt", "r")  # y1
    bool_expressions_results = []  # y2
    print_state(1)
    for line in code:  # x1
        num_line += 1  # y3
        line = line.strip().rstrip()  # y4
        print_state(2)
        if line.startswith("if "):  # x2
            try:  # x3
                expression = get_variable(line)  # y5
                result = calculate_bool_expression(expression)  # y6
                bool_expressions_results.append(int(result))  # y7
            except Exception as e:
                print_error(e)  # y8
        elif "} else {" in line:  # x4
            bool_expressions_results.append(1 -
bool_expressions_results.pop())  # y9
            continue
        elif "}" in line:  # x5
            bool_expressions_results.pop()  # y10
            continue
        elif len(line) == 0 or len(bool_expressions_results) > 0 and not
bool_expressions_results[-1]:  # x6
            continue
        elif " = " in line:  # x7
            splitered_line = line.split(" = ")  # y11
```

```
            print_state(3)
            if len(splitered_line) != 2:  # x8
                print_error("Invalid variable definition")  # y12
            if not line[0][0].isalpha() and "getdata" not in
splitered_line[1] and '"' not in splitered_line[1]:  # x9
                print_error("Variable name must start with a letter")  #
y13
            if "getdata" in splitered_line[1]:  # x10
                variables[splitered_line[0]] = input()  # y14
                continue
            if '"' in splitered_line[1]:  # x11
                variables[splitered_line[0]] =
splitered_line[1].replace('"', "")  # y15
                continue
            e = -1  # y16
            print_state(4)
            try:  # x3
                calc = Calc(splitered_line[1])
                result = calc.calculate()  # y17
            except Exception as er:
                e = er  # y18
            print_state(5)
            if e != -1:  # x12
                try:  # x3
                    result = calculate_bool_expression(splitered_line[1])
# y19
                except:
                    print_error(e)  # y8
            print_state(6)
            variables[splitered_line[0]] = result  # y20
        elif line.startswith("outdata"):  # x13
            try:  # x3
                print(get_value(line))  # y21
            except Exception as e:
                print_error(e)  # y8
        else:
            print_error("Invalid command")  # y22
        print_state(1)
    print_state(0)
main()
```