

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 №3  
«Элементы языка сценариев. Арифметические выражения»

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, вычисляющего значения арифметических выражений в диапазоне  $[-128, 127]$  в целых числах. Автомат выполняется в виде класса, содержащего метод, принимающий на вход выражение, а на выходе выдающий булевское значение, показывающее, является ли выражение корректным. Другим методом реализуемого класса является метод получения результата вычисления.

Выражение в класс передается как строка. Строка может содержать пробелы между операндами, операциями и скобками.

## ТЕКСТОВОЕ ОПИСАНИЕ РЕАЛИЗОВАННОГО АВТОМАТА

На вход программе подается строка – математическое выражение, написанное пользователем в консоли. Далее, математическое выражение отправляется на проверку: верны ли все символы в математическом выражении (скобки, цифры, пробелы и операнды), далее строится обратная польская запись. После формирования обратной польской записи проверяется наличие незакрытых скобок и длина обратной польской записи, если там один элемент, то выводится ошибка. Дальше в случае правильности математического выражения идёт само вычисление обратной польской записи, а также проверки на переполнение операндов и результата операций, а также деления на 0. В случае успеха выводится результат. Если где-то на этапе проверки произошла ошибка, то программа напишет пользователю «Invalid expression». Если где-то на этапе вычисления операнд будет вне диапазона, то программа выведет пользователю «The operand is outside the range: «операнд»». Если результат операции будет вне диапазона, то программа выведет «Overflow: «операнд1 + операнд2 = результат»», и при делении на ноль программа выдаст ошибку «Division by zero».

## ТАБЛИЦЫ ОБОЗНАЧЕНИЙ

Таблица 1. Обозначение входных сигналов.

Входной сигнал	Обозначение
x1	if (not all(char.isdigit() or char in "+-*/" or char.isspace() or char in "()") for char in self.__math_expression)):
x2	for char in self.__math_expression:
x3	if char == "(":
x4	if prev_char != " " and prev_char != "":
x5	elif char == ")":
x6	while stack and stack[-1] != "(":
x7	if not stack:
x8	elif char in "+-*/":
x9	if prev_char != " " and prev_char != "(" and (prev_char != "" and char == "-"):
x10	elif char.isdigit():
x11	if prev_char.isdigit():
x12	if prev_char == '-':
x13	if prev_char != " " and prev_char != "" and prev_char != "(":
x14	if any(char == "(" for char in stack):
x15	if len(reverse_polish_notation) == 1:
x16	while stack:
x17	if not self.__check_expression():
x18	for item in reverse_polish_notation:
x19	if isdigit(item):

Входной сигнал	Обозначение
x20	if num1 > 127 or num1 < -128:
x21	if num2 > 127 or num2 < -128:
x22	if item == "+":
x23	if num1 + num2 > 127 or num1 + num2 < -128:
x24	elif item == "-":
x25	if num1 - num2 > 127 or num1 - num2 < -128:
x26	elif item == "*":
x27	if num1 * num2 > 127 or num1 * num2 < -128:
x28	elif item == "/":
x29	if num2 == 0:

Таблица 2. Обозначение выходных функций.

Выходная функция	Обозначение
y1	print("Note that the calculator performs operations on numbers that are in this range: [-128, 127]!")
y2	math_expression = input("Enter a math expression: ")
y3	calculator = Calc(math_expression)
y4	return False
y5	error_code = "001"
y6	stack = []
y7	reverse_polish_notation = []

Выходная функция	Обозначение
y8	prev_char = ""
y9	stack.append(char)
y10	reverse_polish_notation.append(stack. pop())
y11	stack.pop()
y12	reverse_polish_notation[-1] += char
y13	prev_char = char
y14	reverse_polish_notation.append(stack. pop() + char)
y15	reverse_polish_notation.append(char)
y16	self.__reverse_polish_notation = reverse_polish_notation
y17	return True
y18	return "Invalid expression"
y19	reverse_polish_notation = self.__reverse_polish_notation
y20	stack.append(int(item))
y21	num2 = stack.pop()
y22	num1 = stack.pop()
y23	return "The operand is outside the range: " + str(num1)
y24	return "The operand is outside the range: " + str(num2)
y25	return f"Overflow: {num1} + {num2} = {num1 + num2}"
y26	stack.append(num1 + num2)

Выходная функция	Обозначение
y27	return f"Overflow: {num1} - {num2} = {num1 - num2}"
y28	stack.append(num1 - num2)
y29	return f"Overflow: {num1} * {num2} = {num1 * num2}"
y30	stack.append(num1 * num2)
y31	return "Division by zero"
y32	stack.append(num1 // num2)
y33	return f"Result: {stack[-1]}"

## БЛОК-СХЕМА

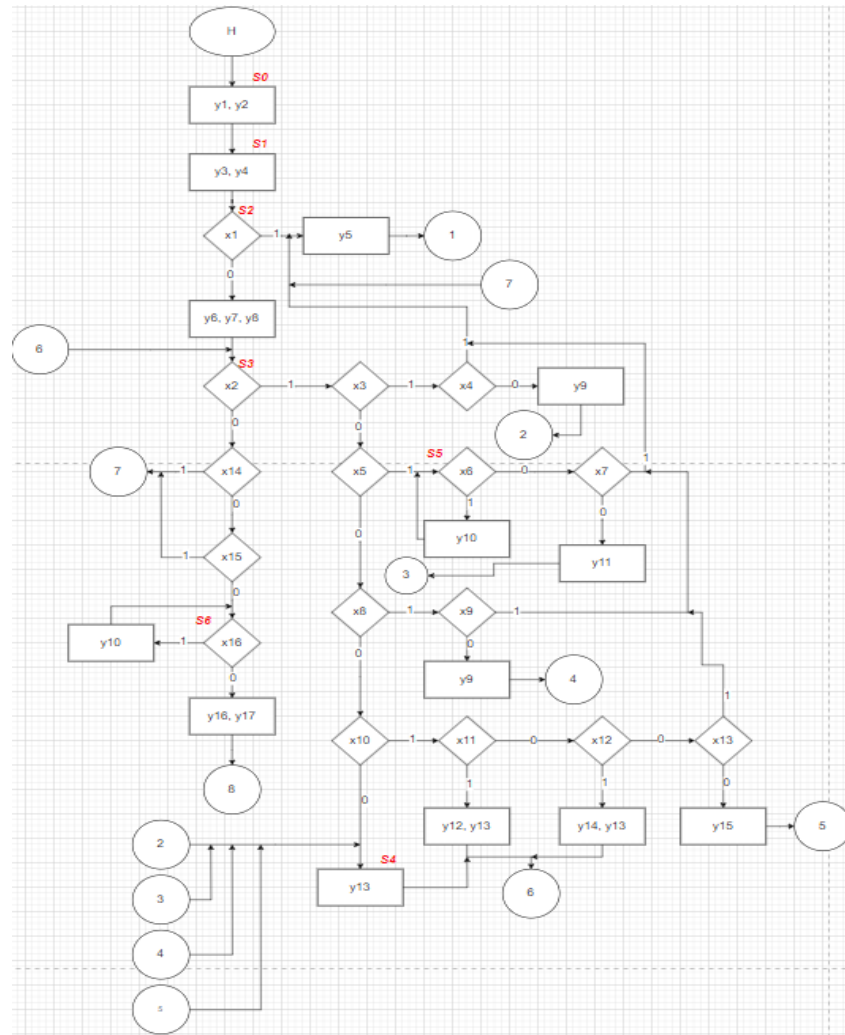


Рис 1.1. Блок-схема автомата типа Мили – main и метод проверки математического выражения.



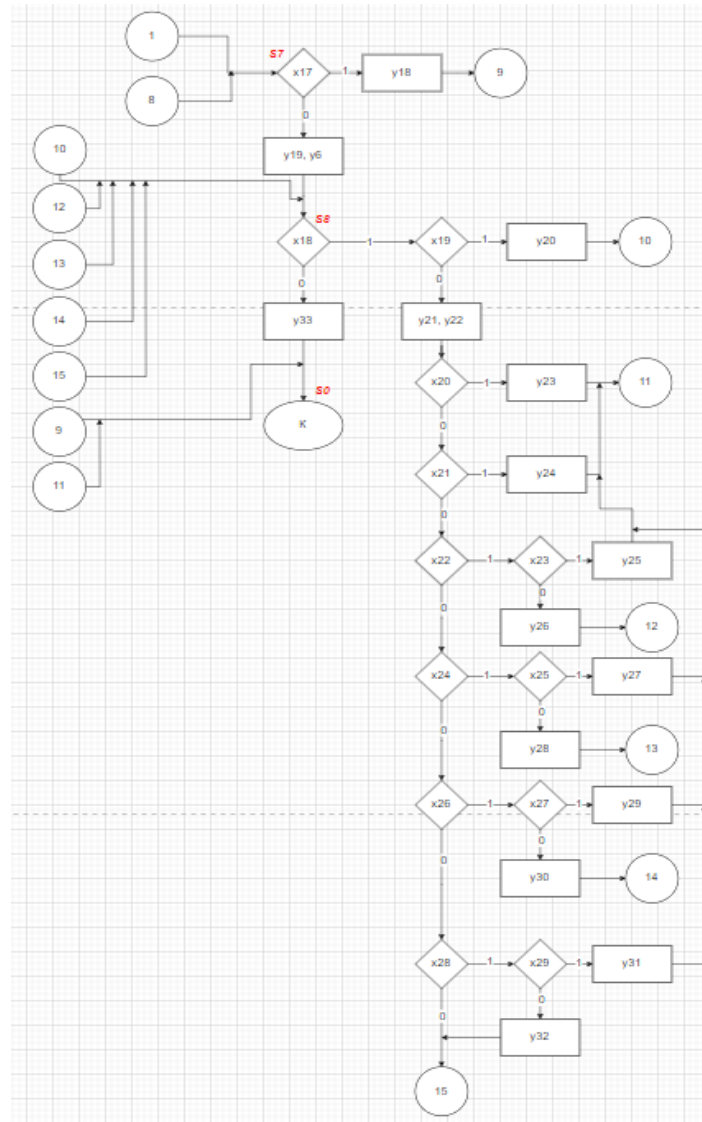


Рис 1.2. Блок-схема автомата типа Мили – метод вычисления математического выражения

## ГРАФ АВТОМАТА

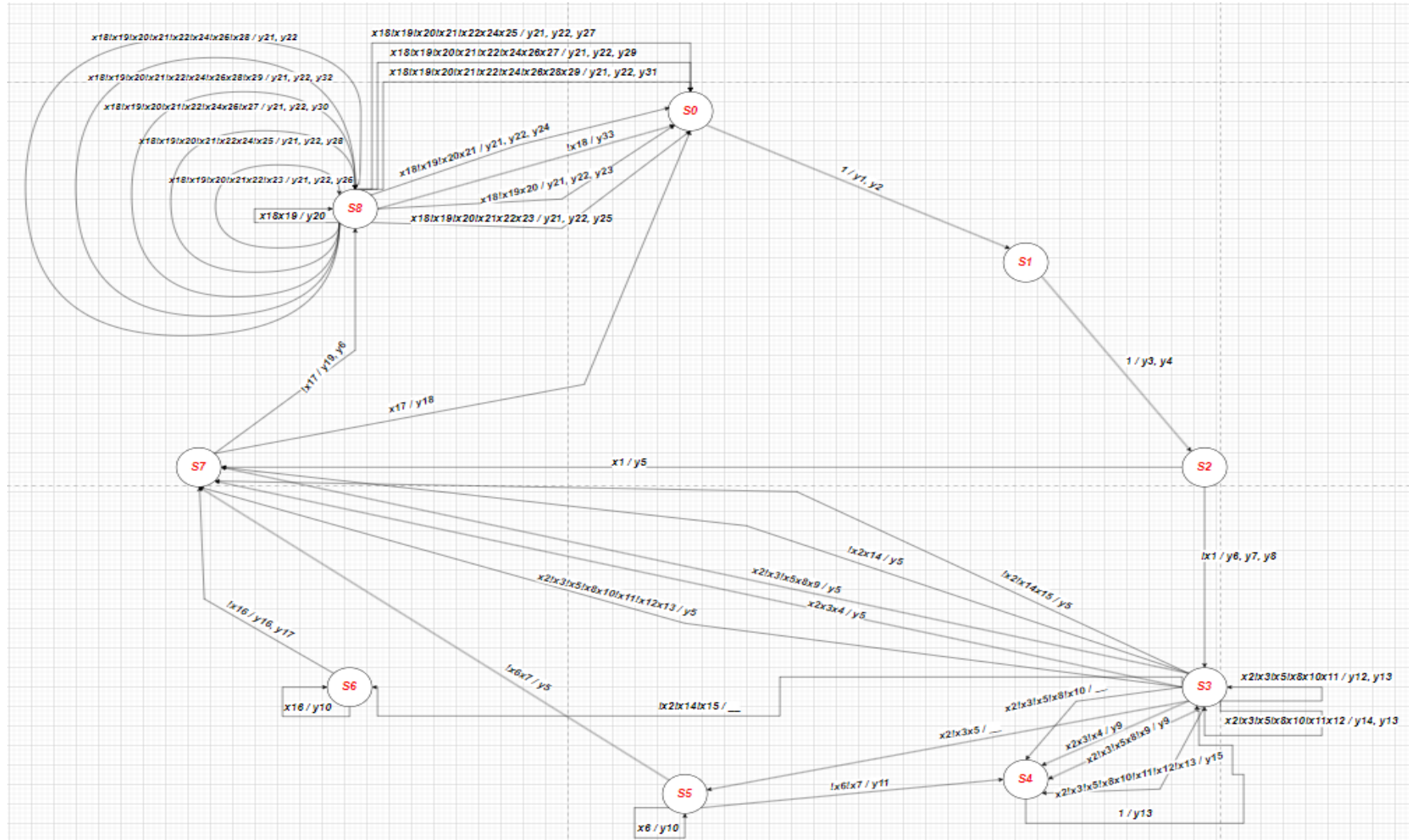


Рис 2. Граф автомата Мили для проверки и подсчёта математического выражения.

## ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

```
Note that the calculator performs operations on numbers that are in this range: [-128, 127]!  
Enter a math expression: (55 + (43))  
Result: 98
```

Рис 3.1. Тест и вывод программы без ошибок

out\_states.txt – Блокнот

Файл Правка Формат Вид Справка

```
S0 -> S1 -> S2 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S5 -> S4 -> S7 -> S8 -> S8 -> S0
```

Рис 3.2. Состояния программы без ошибок

```
Note that the calculator performs operations on numbers that are in this range: [-128, 127]!  
Enter a math expression: (55 + ((43))  
Invalid expression
```

Рис 4.1. Лишняя скобка в вводе

Calc.py main.py out\_states.txt x

```
1 S0 -> S1 -> S2 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S7 -> S0
```

Рис 4.2. Состояния программы с вводом лишней скобки

```
Note that the calculator performs operations on numbers that are in this range: [-128, 127]!  
Enter a math expression: 2 ++ 2  
Invalid expression
```

Рис 5.1. Два знака подряд в вводе

Calc.py main.py out\_states.txt x

```
1 S0 -> S1 -> S2 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S3 -> S7 -> S0
```

Рис 5.2. Состояния программы с вводом двух знаков подряд

```
Note that the calculator performs operations on numbers that are in this range: [-128, 127]!  
Enter a math expression: 128 + 1  
The operand is outside the range: 128
```

Рис 6.1.1. Операнд вне диапазона

Calc.py main.py out\_states.txt x

```
1 S0 -> S1 -> S2 -> S3 -> S4 -> S3 -> S3 -> S4 -> S3 -> S4 -> S3 -> S4 -> S6 -> S7 -> S8 -> S8 -> S0
```

Рис 6.2.1. Состояния программы с вводом операнда вне диапазона

```
Note that the calculator performs operations on numbers that are in this range: [-128, 127]!  
Enter a math expression: -129 + 1  
The operand is outside the range: -129
```

Рис 6.1.2 Операнд вне диапазона

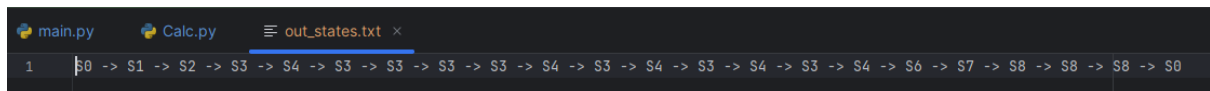


Рис 6.2.2. Состояния программы с вводом операнда вне диапазона

```
Note that the calculator performs operations on numbers that are in this range: [-128, 127]!  
Enter a math expression: 65 * 2  
Overflow: 65 * 2 = 130
```

Рис 7.1. Переполнение в результате операции

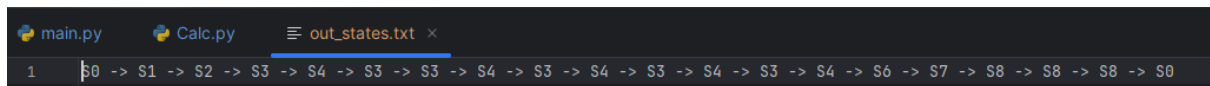


Рис 7.2. Состояния программы с переполнение в результате операции

## **ИТОГИ**

По итогу данной работы был спроектирован и реализован автомат типа Мили для подсчёта простейших математических выражений со всеми операциями и скобочными выражениями в целых числа и в диапазоне [-128, 127]. Алгоритм автомата был реализован при помощи ООП, а точнее с помощью класса и двух методов – проверка и подсчёт. Также были построены блок-схема и граф-схема алгоритма.

## ПРИЛОЖЕНИЕ

### Основной файл:

```
from Calc import *
#Главная функция
def main():
    print_state("0")
    # Печатаем предупреждение
    print("Note that the calculator performs operations on numbers that are
in this range: [-128, 127]!") #y1
    # Вводим выражение
    math_expression = input("Enter a math expression: ") # y2
    print_state(1)
    # Создаем объект класса Calc
    caculator = Calc(math_expression) # y3
    # Выводим результат вычисления
    print(caculator.calculate()) # y4
    print_state(0)
#Запускаем главную функцию
main()
```

### Файл класса:

```
from typing import Any
# Функция для проверки является ли строка числом
def isdigit(num: str) -> bool:
    try:
        int(num)
        return True
    except ValueError:
        return False
out_states = open("out_states.txt", "w")
def print_state(state):
    print(f"S{state}{" " if state == 0 else " -> "}", file=out_states,
end="")
class Calc:
    # Приватные поля класса
    __math_expression = "" # Само выражение в виде строки
    __reverse_polish_notation = [] # Обратная польская запись выражения
    # Конструктор класса
    def __init__(self, math_expression):
        self.__math_expression = math_expression
    # Приватный метод для проверки корректности выражения
    def __check_expression(self) -> bool:
        # Удостоверимся, что в выражении нет недопустимых символов
        print_state(2)
        if (not all(char.isdigit() or char in "+-*/" or char.isspace() or
char in "()" # x1
for char in self.__math_expression)):
            return False # y5
        # Создаем стек и список для обратной польской записи выражения
        # и строку для хранения предыдущего символа
        stack = [] # y6
        reverse_polish_notation = [] # y7
        prev_char = "" # y8
        # Удостоверимся, что скобки, операторы и числа расположены
        правильно
```

```

for char in self.__math_expression: # x2
    print_state(3)
    # Если символ - скобка, то добавляем ее в стек
    if char == "(": # x3
        # Если перед скобкой не было пробела, то возвращаем False
        if prev_char != " " and prev_char != "": # x4
            return False # y5
        stack.append(char) # y9
    # Если символ - закрывающая скобка,
    elif char == ")": # x5
        # Переносим все операторы в обратную польскую запись
        # пока не встретим открывающую скобку
        while stack and stack[-1] != "(": # x6
            print_state(5)
            reverse_polish_notation.append(stack.pop()) # y10
        # Вновь проверяем стек на пустоту т.к. в нем могут быть
        только операторы
        # перед закрывающей скобкой
        if not stack: # x7
            return False # y5
        # Удаляем из стека открывающую скобку
        stack.pop() # y11
    # Если символ - оператор, то добавляем его в стек
    elif char in "+-*/": # x8
        # Если перед оператором ничего, то возвращаем False
        if prev_char != " " and prev_char != "(" and (prev_char !=
        "" and char == "-"): # x9
            return False # y5
        stack.append(char) # y9
    # Если символ - цифра, то добавляем ее в обратную польскую
    запись
    elif char.isdigit(): # x10
        # Если перед цифрой тоже цифра, то добавляем ее к
        предыдущей цифре
        if prev_char.isdigit(): # x11
            reverse_polish_notation[-1] += char # y12
            prev_char = char # y13
            continue
        # Если перед цифрой знак минус, то добавляем его к цифре
        if prev_char == '-': # x12
            reverse_polish_notation.append(stack.pop() + char) #
        y14
            prev_char = char # y13
            continue
        # Если перед цифрой не было пробела, то возвращаем False
        if prev_char != " " and prev_char != "" and prev_char !=
        "(": # x 13
            return False # y5
            reverse_polish_notation.append(char) # y15
        # Обновляем предыдущий символ
        print_state(4)
        prev_char = char # y13
    # Проверяем остались ли в стеке не закрытые скобки
    # Если остались, то возвращаем False
    if any(char == "(" for char in stack): # x14
        return False # y5

```

```

# Если в обратной польской записи всего 1 число, то возвращаем
False
if len(reverse_polish_notation) == 1: # x15
    return False # y5
# Переносим все операторы в обратную польскую запись
while stack: # x16
    print_state(6)
    reverse_polish_notation.append(stack.pop()) # y10
self.__reverse_polish_notation = reverse_polish_notation # y16
return True # y17
# Публичный метод вычисления результата заданного выражения
def calculate(self) -> str | int | Any:
    # Если выражение некорректно, то возвращаем "Invalid expression"
    if not self.__check_expression(): # x17
        print_state(7)
        return "Invalid expression" # y18
    print_state(7)
    # Два вывода одного и того же состояния, сделанные выше, сделаны
    # для того, чтобы в случае попадания
    # в if состояние вывелось, и в случае не попадания в if оно тоже
    # вывелось.
    # Если же поставить вывод состояния перед if, то S7 выведется
    # раньше чем нужно.
    # Грубо говоря, - это костыль
    reverse_polish_notation = self.__reverse_polish_notation # y19
    stack = [] # y6
    # Вычисляем обратную польскую запись
    for item in reverse_polish_notation: # x18
        print_state(8)
        # Если элемент - число, то добавляем его в стек
        if isdigit(item): # x19
            stack.append(int(item)) # y20
        # Если элемент - оператор, то вычисляем результат
        else:
            # Достаем из стека 2 числа
            num2 = stack.pop() # y21
            num1 = stack.pop() # y22
            # Если числа выходят за пределы [-128, 127], то возвращаем
            "Overflow"
            if num1 > 127 or num1 < -128: # x20
                return "The operand is outside the range: " + str(num1)
            # y23
            if num2 > 127 or num2 < -128: # x21
                return "The operand is outside the range: " + str(num2)
            # y24
            # Проводим операцию в зависимости от оператора
            # Если результат выходит за пределы [-128, 127], то
            # возвращаем "Overflow"
            if item == "+": # x22
                if num1 + num2 > 127 or num1 + num2 < -128: # x23
                    return f"Overflow: {num1} + {num2} = {num1 + num2}"
            # y25
                stack.append(num1 + num2) # y26
            elif item == "-": # x24
                if num1 - num2 > 127 or num1 - num2 < -128: # x25
                    return f"Overflow: {num1} - {num2} = {num1 - num2}"
            # y27

```



```

        stack.append(num1 - num2)  # y28
    elif item == "*":  # x26
        if num1 * num2 > 127 or num1 * num2 < -128:  # x 27
            return f"Overflow: {num1} * {num2} = {num1 * num2}"
# y29
        stack.append(num1 * num2)  # y30
    elif item == "/":  # x28
        if num2 == 0:  # x29
            return "Division by zero"  # y31
        stack.append(num1 // num2)  # y32
    # Результат лежит в последнем элементе стека - достаём его и
    # возвращаем
    return f"Result: {stack[-1]}"  # y33

```