Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Программная инженерия
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №7

Исчисления и абстрактная интерпретация
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата — инициалы, фамилия

Студент КИ23-16/1б, 32322546 _____ Е. А. Гуртякин
номер группы, зачётной книжки — подпись, дата — инициалы, фамилия

Красноярск 2025

1.1 Цель

Исследование проблем вычислимости без использования абстрактной машины Тьюринга.

1.2 Задачи

1. Ознакомиться с теоретическими сведениями по проблемам вычислимости и разрешимости, а также метода абстрактной интерпретации.
2. Получить у преподавателя собственный вариант задания, предусматривающего построение вычислителя заданной функции над целыми числами (1 часть), а также проведение абстрактной интерпретации (2 часть).
3. Используя изученные механизмы, произвести программную реализацию вычислителя заданной математической функции для заданных аргументов, причем исключительно средствами примитивной и частичной рекурсии. Области определения и значений не обязательно совпадают с множеством натуральных чисел, могут быть ограничены "компьютерными числами" (например, Integer.MAX_VALUE и Integer.MIN_VALUE). Это следует учитывать при конструировании вычислителя.
4. Используя метод абстрактной интерпретации, для произвольной (на выбор и вкус студента) подпрограммы определить знаки всех переменных. Минимальное количество строк кода процедуры без учета комментариев не менее 10.
5. Написать отчет и представить его к защите вместе с исходным кодом программы. Защита может проводиться как в аудитории, так и в исключительных случаях дистанционно.

1.3 Задание

В части 1 необходимо произвести программную реализацию вычислителя заданной математической функции для заданных аргументов, причем исключительно средствами примитивной и частичной рекурсии, или

формально доказать невозможность этого. Привести примеры выполнения вычислений.

В части 2 необходимо, используя метод абстрактной интерпретации, для произвольной программной процедуры с количеством строк кода без комментариев не менее 10, определить знаки всех переменных.

Мною был получен вариант 2:

f(x, y) = x ^ y, где ^ — это операция возведения в степень, а $y \geq 0$.

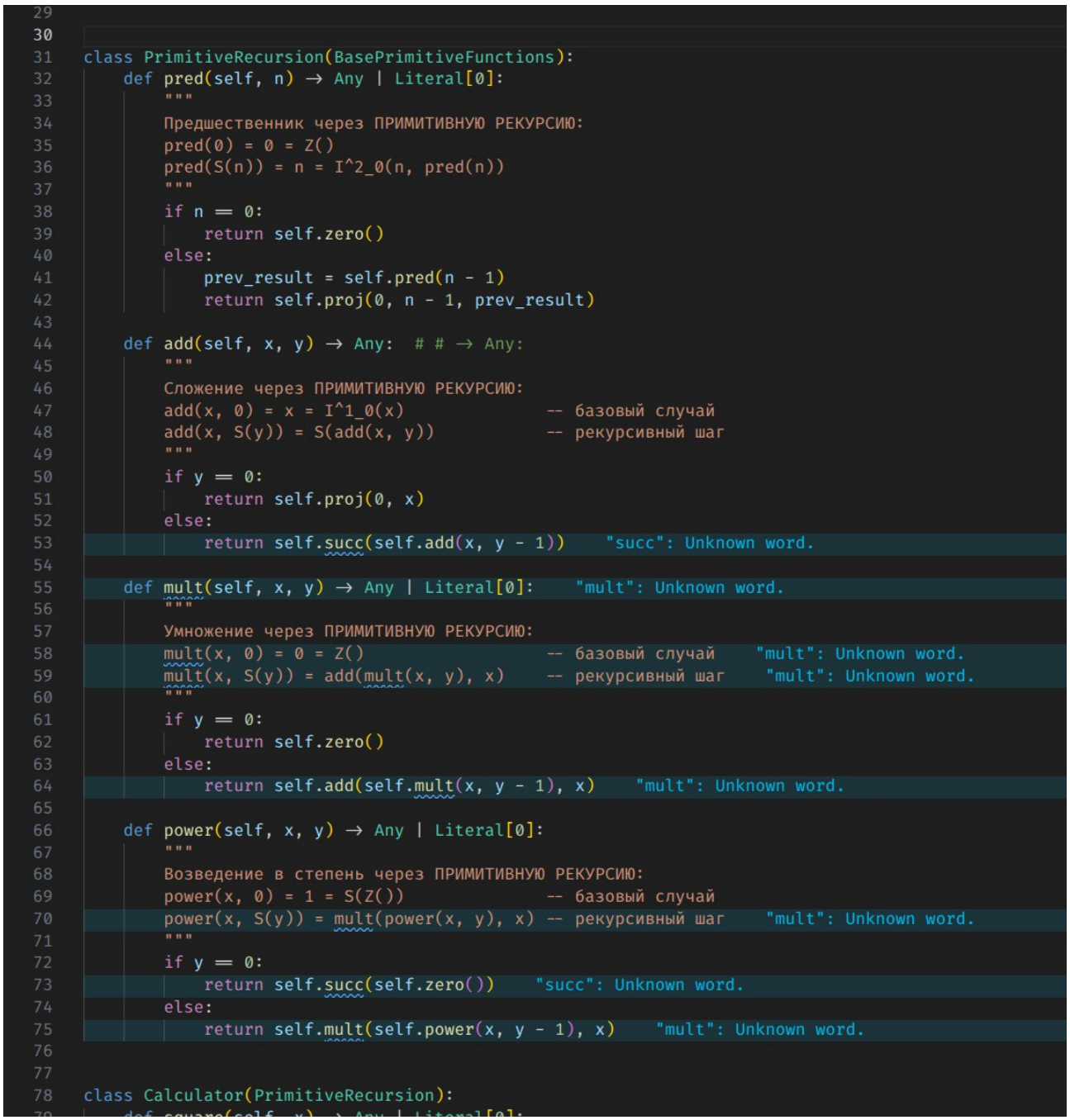
2 ХОД РАБОТЫ

2.1 Задание 1

Программная реализация вычислителя функции для моего варианта представлена на рисунках 1-3.

```python
class BasePrimitiveFunctions:
    @staticmethod
    def zero(*args) -> Literal[0]:
        """
        Нулевая функция: Z() = 0 или O^n(x_1, ... ,x_n) = 0
        Возвращает 0 для любых аргументов
        """
        return 0

    @staticmethod
    def succ(n) -> Any:    "succ": Unknown word.
        """
        Функция следования: S(n) = n + 1
        """
        return n + 1

    @staticmethod
    def proj(k, *args) -> Any:
        """
        Функция проекции: I^n_k(x_1, ... , x_n) = x_k
        Возвращает k-й аргумент

        Пример: proj(1, a, b, c) = b
        """
        return args[k]


class PrimitiveRecursion(BasePrimitiveFunctions):
    def pred(self, n) -> Any | Literal[0]:
        """
        Предшественник через ПРИМИТИВНУЮ РЕКУРСИЮ:
        pred(0) = 0 = Z()
        pred(S(n)) = n = I^2_0(n, pred(n))
        """
        if n == 0:
            return self.zero()
        else:
            prev_result = self.pred(n - 1)
            return self.proj(0, n - 1, prev_result)

    def add(self, x, y) -> Any:  # # -> Any:
        """
        Сложение через ПРИМИТИВНУЮ РЕКУРСИЮ:
        add(x, 0) = x = I^1_0(x)              -- базовый случай
        add(x, S(y)) = S(add(x, y))           -- рекурсивный шаг
        """
        if y == 0:
            return self.proj(0, x)
        else:
            return self.succ(self.add(x, y - 1))    "succ": Unknown word.
```

Рисунок 1 - Программная реализация вычислителя функции, часть 1



```
class PrimitiveRecursion(BasePrimitiveFunctions):
    def pred(self, n) → Any | Literal[0]:
        """
        Предшественник через ПРИМИТИВНУЮ РЕКУРСИЮ:
        pred(0) = 0 = Z()
        pred(S(n)) = n = I^2_0(n, pred(n))
        """
        if n == 0:
            return self.zero()
        else:
            prev_result = self.pred(n - 1)
            return self.proj(0, n - 1, prev_result)

    def add(self, x, y) → Any:  # # → Any:
        """
        Сложение через ПРИМИТИВНУЮ РЕКУРСИЮ:
        add(x, 0) = x = I^1_0(x)              -- базовый случай
        add(x, S(y)) = S(add(x, y))           -- рекурсивный шаг
        """
        if y == 0:
            return self.proj(0, x)
        else:
            return self.succ(self.add(x, y - 1))    "succ": Unknown word.

    def mult(self, x, y) → Any | Literal[0]:    "mult": Unknown word.
        """
        Умножение через ПРИМИТИВНУЮ РЕКУРСИЮ:
        mult(x, 0) = 0 = Z()                  -- базовый случай    "mult": Unknown word.
        mult(x, S(y)) = add(mult(x, y), x)    -- рекурсивный шаг    "mult": Unknown word.
        """
        if y == 0:
            return self.zero()
        else:
            return self.add(self.mult(x, y - 1), x)    "mult": Unknown word.

    def power(self, x, y) → Any | Literal[0]:
        """
        Возведение в степень через ПРИМИТИВНУЮ РЕКУРСИЮ:
        power(x, 0) = 1 = S(Z())              -- базовый случай
        power(x, S(y)) = mult(power(x, y), x) -- рекурсивный шаг    "mult": Unknown word.
        """
        if y == 0:
            return self.succ(self.zero())    "succ": Unknown word.
        else:
            return self.mult(self.power(x, y - 1), x)    "mult": Unknown word.


class Calculator(PrimitiveRecursion):
```

Рисунок 2 - Программная реализация вычислителя функции, часть 2

```python
class Calculator(PrimitiveRecursion):
    def square(self, x) -> Any | Literal[0]:
        """
        g(x) = x² через СУПЕРПОЗИЦИЮ
        """
        return self.mult(self.proj(0, x), self.proj(0, x))    "mult": Un

    def power_self(self, x) -> Any | Literal[0]:
        """
        f(x) = x^x через СУПЕРПОЗИЦИЮ
        """
        return self.power(self.proj(0, x), self.proj(0, x))

    def cube(self, x) -> Any | Literal[0]:
        """
        c(x) = x³ через СУПЕРПОЗИЦИЮ
        """
        return self.mult(self.square(x), self.proj(0, x))    "mult": Unk


if __name__ == "__main__":
    calc = Calculator()
    print("Вычислитель x^y")
    examples = [(1, 3), (4, 2), (2, 4)]
    for x, y in examples:
        result = calc.power(x, y)
        print(f"  {x:>4} ^ {y:<4} = {result:>10}")

    print("Вычислитель x^x")
    for x in (1, 2, 3, 5):
        result = calc.power_self(x)
        print(f"  {x:>4} ^ {x:<4} = {result:>10}")
```

Рисунок 3 - Программная реализация вычислителя функции, часть 3

Пример работы вычислителя представлен на рисунке 4.

```
(vuz-raboti-py3.13) master@FedoraDesktop:~/Work/Vuz-raboti/5th_semester/теория автоматов/pr7/pr7Gurtyakin$ python recursion.py
Вычислитель x^y
     1 ^ 3    =          1
     4 ^ 2    =         16
     2 ^ 4    =         16
Вычислитель x^x
     1 ^ 1    =          1
     2 ^ 2    =          4
     3 ^ 3    =         27
     5 ^ 5    =       3125
```

Рисунок 4 - Пример работы вычислителя

2.2 Часть 2

Необходимо, используя метод абстрактной интерпретации, для произвольной программной процедуры с количеством строк кода без комментариев не менее 10, определить знаки всех переменных.

Код функции и анализатора, представлены на рисунках 5-8.

```python
from enum import Enum
from typing import Dict, List, Tuple


def function(a, b) -> Any:
    c = a + b
    d = a * c
    e = b - a
    f = c * d
    g = a + d
    h = e * f
    i = g - b
    j = h + i
    k = d * e
    x = j * k
    t = x * a
    return x + t


class Sign(Enum):
    """
    Абстрактная область знаков для абстрактной интерпретации.
    """

    UNKNOWN = "UNKNOWN"  # Неопределенное
    NEGATIVE = "-"  # Отрицательное
    ZERO = "0"  # Ноль
    POSITIVE = "+"  # Положительное


class SignAnalysis:
    """
    Абстрактная интерпретация для анализа знаков переменных.
    """

    @staticmethod
    def abstract(value: int) -> Sign:
        """Абстракция конкретного значения"""
        if value < 0:
            return Sign.NEGATIVE
        elif value == 0:
            return Sign.ZERO
        else:
            return Sign.POSITIVE
```

Рисунок 5 - Исходный код анализируемой функции

```python
class SignAnalysis:
    """
    Абстрактная интерпретация для анализа знаков переменных.
    """

    @staticmethod
    def abstract(value: int) -> Sign:
        """Абстракция конкретного значения"""
        if value < 0:
            return Sign.NEGATIVE
        elif value == 0:
            return Sign.ZERO
        else:
            return Sign.POSITIVE

    @staticmethod
    def join(s1: Sign, s2: Sign) -> Sign:
        """Операция объединения (join) в решетке знаков."""
        if s1 == s2:
            return s1
        if s1 == Sign.UNKNOWN:
            return s2
        if s2 == Sign.UNKNOWN:
            return s1
        return Sign.UNKNOWN

    @staticmethod
    def add(s1: Sign, s2: Sign) -> Sign:
        """Абстрактное сложение"""
        if s1 == Sign.UNKNOWN or s2 == Sign.UNKNOWN:
            return Sign.UNKNOWN
        if s1 == Sign.ZERO:
            return s2
        if s2 == Sign.ZERO:
            return s1
        if s1 == s2:
            return s1
        return Sign.UNKNOWN

    @staticmethod
    def sub(s1: Sign, s2: Sign) -> Sign:
        """Абстрактное вычитание"""
        if s1 == Sign.UNKNOWN or s2 == Sign.UNKNOWN:
            return Sign.UNKNOWN

        neg_s2 = SignAnalysis.negate(s2)
        return SignAnalysis.add(s1, neg_s2)

    @staticmethod
    def mult(s1: Sign, s2: Sign) -> Sign:    "mult": Unknown word.
        """Абстрактное умножение"""
        if s1 == Sign.UNKNOWN or s2 == Sign.UNKNOWN:
```

Git Graph

Рисунок 6 - Исходный код программы

```python
class AbstractInterpreter:
    """
    Абстрактный интерпретатор для анализа знаков переменных в программе.
    """

    def __init__(self) -> None:
        self.variables: Dict[str, Sign] = {}
        self.trace: List[Tuple[int, str, str, Sign]] = []

    def set_var(self, line: int, name: str, expr: str, sign: Sign) -> None:
        """Установить абстрактное значение переменной и записать в путь трассировки."""
        self.variables[name] = sign
        self.trace.append((line, name, expr, sign))

    def get_var(self, name: str) -> Sign:
        """Получить абстрактное значение переменной"""
        return self.variables.get(name, Sign.UNKNOWN)

    def analyze_function(self, a_sign: Sign, b_sign: Sign) -> Dict[str, Sign]:
        """
        Абстрактная интерпретация функции some_function.

        Анализируемая программа:

        def function(a, b):
            c = a + b
            d = a * c
            e = b - a
            f = c * d
            g = a + d
            h = e * f
            i = g - b
            j = h + i
            k = d * e
            x = j * k
            t = x * a
            return x + t
        """
        self.variables = {}
        self.trace = []

        self.variables["a"] = a_sign
        self.variables["b"] = b_sign

        c = SignAnalysis.add(self.get_var("a"), self.get_var("b"))
        self.set_var(1, "c", "a + b", c)

        d = SignAnalysis.mult(self.get_var("a"), self.get_var("c"))    "mult": Unknown word.
        self.set_var(2, "d", "a * c", d)

        e = SignAnalysis.sub(self.get_var("b"), self.get_var("a"))
        self.set_var(3, "e", "b - a", e)
```

Рисунок 7 - Исходный код программы

```
class AbstractInterpreter:
    def analyze_function(self, a_sign: Sign, b_sign: Sign) → Dict[str, Sign]:
        Абстрактная интерпретация функции some_function.

        Анализируемая программа:

        def function(a, b):
            c = a + b
            d = a * c
            e = b - a
            f = c * d
            g = a + d
            h = e * f
            i = g - b
            j = h + i
            k = d * e
            x = j * k
            t = x * a
            return x + t
        """
        self.variables = {}
        self.trace = []

        self.variables["a"] = a_sign
        self.variables["b"] = b_sign

        c = SignAnalysis.add(self.get_var("a"), self.get_var("b"))
        self.set_var(1, "c", "a + b", c)

        d = SignAnalysis.mult(self.get_var("a"), self.get_var("c"))    "mult": Unknown word.
        self.set_var(2, "d", "a * c", d)

        e = SignAnalysis.sub(self.get_var("b"), self.get_var("a"))
        self.set_var(3, "e", "b - a", e)

        f = SignAnalysis.mult(self.get_var("c"), self.get_var("d"))    "mult": Unknown word.
        self.set_var(4, "f", "c * d", f)

        g = SignAnalysis.add(self.get_var("a"), self.get_var("d"))
        self.set_var(5, "g", "a + d", g)

        h = SignAnalysis.mult(self.get_var("e"), self.get_var("f"))    "mult": Unknown word.
        self.set_var(6, "h", "e * f", h)

        i = SignAnalysis.sub(self.get_var("g"), self.get_var("b"))
        self.set_var(7, "i", "g - b", i)

        j = SignAnalysis.add(self.get_var("h"), self.get_var("i"))
        self.set_var(8, "j", "h + i", j)

        k = SignAnalysis.mult(self.get_var("d"), self.get_var("e"))    "mult": Unknown word.
        self.set_var(9, "k", "d * e", k)
Graph
```

Рисунок 8 – Код программы

Условные обозначения, использованные при анализе:

+: переменная принимает значения строго больше нуля

−: переменная принимает значения строго меньше нуля

0: переменная принимает только значение 0

Unknown: переменная принимает любые значения {−;0;+}

Анализ знаков переменных при a: +, b: + рисунок 9

```
Итоговые знаки переменных:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  a          = +       (положительный)
  b          = +       (положительный)
  c          = +       (положительный)
  d          = +       (положительный)
  e          = UNKNOWN (неизвестен)
  f          = +       (положительный)
  g          = +       (положительный)
  h          = UNKNOWN (неизвестен)
  i          = UNKNOWN (неизвестен)
  j          = UNKNOWN (неизвестен)
  k          = UNKNOWN (неизвестен)
  x          = UNKNOWN (неизвестен)
  t          = UNKNOWN (неизвестен)
  result     = UNKNOWN (неизвестен)
```

Рисунок 9 – Итоговые знаки переменных

Анализ знаков переменных при a: +, b: - рисунок 10

```
Итоговые знаки переменных:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  a          = +       (положительный)
  b          = -       (отрицательный)
  c          = UNKNOWN (неизвестен)
  d          = UNKNOWN (неизвестен)
  e          = -       (отрицательный)
  f          = UNKNOWN (неизвестен)
  g          = UNKNOWN (неизвестен)
  h          = UNKNOWN (неизвестен)
  i          = UNKNOWN (неизвестен)
  j          = UNKNOWN (неизвестен)
  k          = UNKNOWN (неизвестен)
  x          = UNKNOWN (неизвестен)
  t          = UNKNOWN (неизвестен)
  result     = UNKNOWN (неизвестен)
```

Рисунок 10 – Итоговые знаки переменных

Анализ знаков переменных при a: -, b: + рисунок 11

```
Итоговые знаки переменных:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  a          = -      (отрицательный)
  b          = +      (положительный)
  c          = UNKNOWN (неизвестен)
  d          = UNKNOWN (неизвестен)
  e          = +      (положительный)
  f          = UNKNOWN (неизвестен)
  g          = UNKNOWN (неизвестен)
  h          = UNKNOWN (неизвестен)
  i          = UNKNOWN (неизвестен)
  j          = UNKNOWN (неизвестен)
  k          = UNKNOWN (неизвестен)
  x          = UNKNOWN (неизвестен)
  t          = UNKNOWN (неизвестен)
  result     = UNKNOWN (неизвестен)
```

Рисунок 11 – Итоговые знаки переменных

Анализ знаков переменных при a: -, b: - рисунок 12

```
Итоговые знаки переменных:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  a          = -      (отрицательный)
  b          = -      (отрицательный)
  c          = -      (отрицательный)
  d          = +      (положительный)
  e          = UNKNOWN (неизвестен)
  f          = -      (отрицательный)
  g          = UNKNOWN (неизвестен)
  h          = UNKNOWN (неизвестен)
  i          = UNKNOWN (неизвестен)
  j          = UNKNOWN (неизвестен)
  k          = UNKNOWN (неизвестен)
  x          = UNKNOWN (неизвестен)
  t          = UNKNOWN (неизвестен)
  result     = UNKNOWN (неизвестен)
```

Рисунок 12 – Итоговые знаки переменных

Анализ знаков переменных при a: 0, b: + рисунок 13

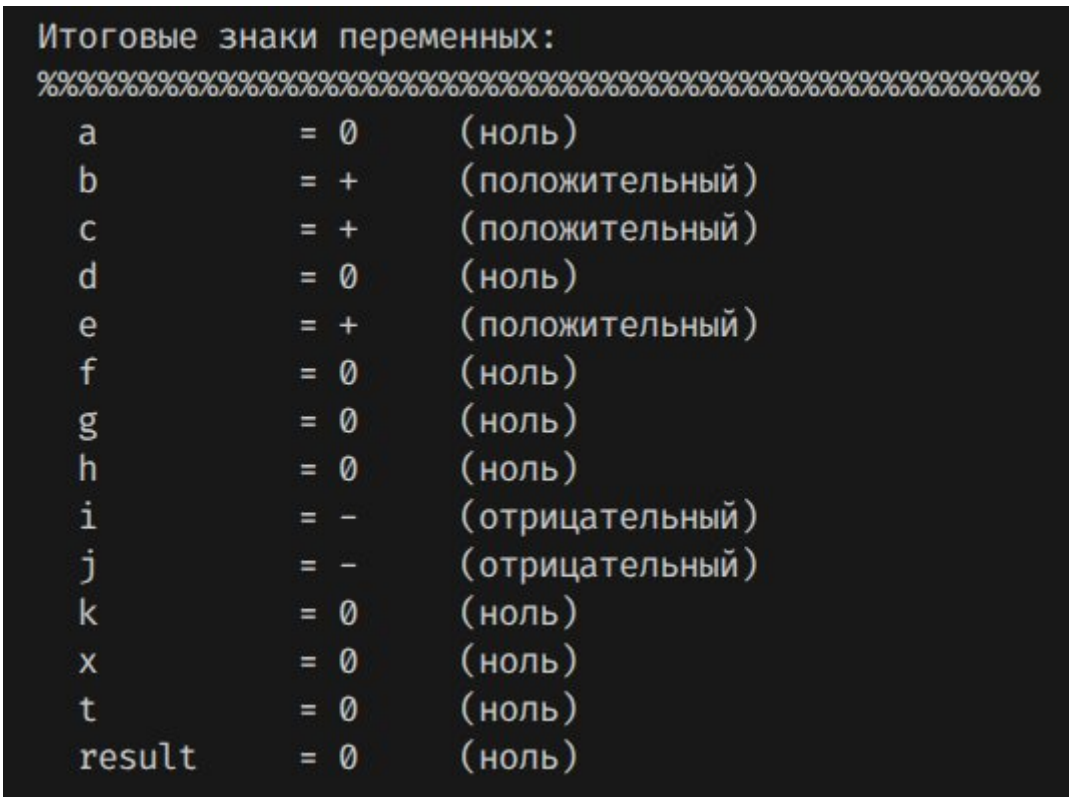


Рисунок 13 – Итоговые знаки переменных

Анализ знаков переменных при a: +, b: 0 рисунок 14

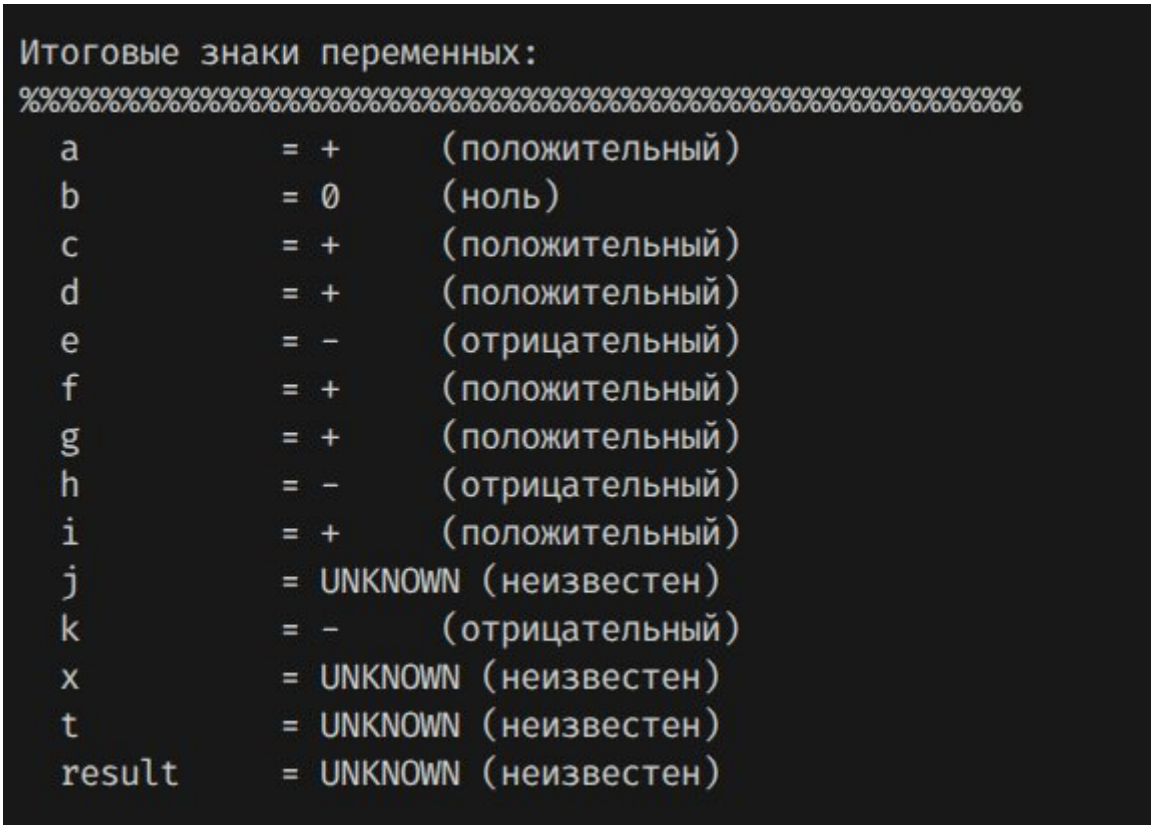


Рисунок 14 – Итоговые знаки переменных

Анализ знаков переменных при a: 0, b: - рисунок 15

```
Итоговые знаки переменных:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  a          = 0     (ноль)
  b          = -     (отрицательный)
  c          = -     (отрицательный)
  d          = 0     (ноль)
  e          = -     (отрицательный)
  f          = 0     (ноль)
  g          = 0     (ноль)
  h          = 0     (ноль)
  i          = +     (положительный)
  j          = +     (положительный)
  k          = 0     (ноль)
  x          = 0     (ноль)
  t          = 0     (ноль)
  result     = 0     (ноль)
```

Рисунок 15 – Итоговые знаки переменных

Анализ знаков переменных при a: -, b: 0 рисунок 16

```
Итоговые знаки переменных:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  a          = -     (отрицательный)
  b          = 0     (ноль)
  c          = -     (отрицательный)
  d          = +     (положительный)
  e          = +     (положительный)
  f          = -     (отрицательный)
  g          = UNKNOWN (неизвестен)
  h          = -     (отрицательный)
  i          = UNKNOWN (неизвестен)
  j          = UNKNOWN (неизвестен)
  k          = +     (положительный)
  x          = UNKNOWN (неизвестен)
  t          = UNKNOWN (неизвестен)
  result     = UNKNOWN (неизвестен)
```

Рисунок 16 – Итоговые знаки переменных

Анализ знаков переменных при a: 0, b: 0 рисунок 17

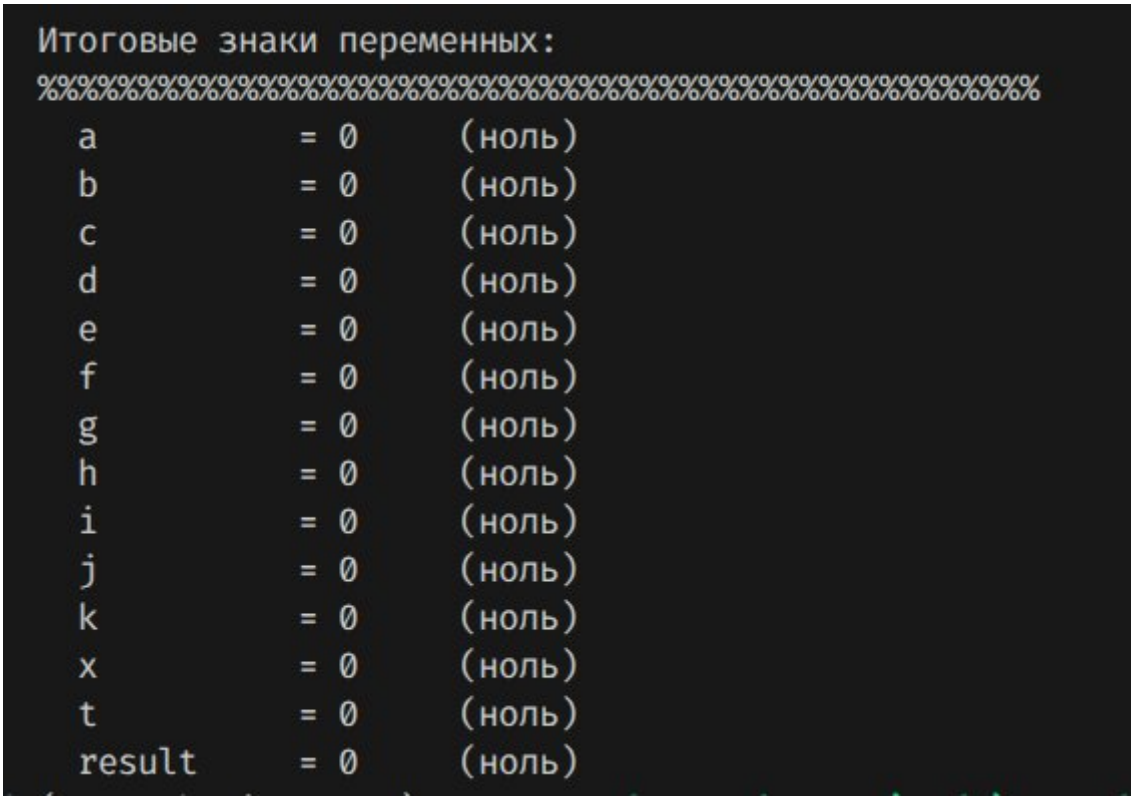


Рисунок 17 – Итоговые знаки переменных

Вывод

По результатам работы был изучен теоретический материал по теме «Исчисления и абстрактная интерпретация». Все поставленные цели и задачи были выполнены. Задания были выполнены и помогли лучше усвоить пройденный материал.