

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Математический факультет
Кафедра функционального анализа

Отчет по дисциплине:
«Программирование криптографических алгоритмов»

Направление 02.04.01 Математика и компьютерные науки

| | | | |
|---------------|----------------|-----------|------------------|
| Преподаватель | _____ | к.ф.-м.н. | М.Г. Завгородний |
| | <i>подпись</i> | | |
| Обучающийся | _____ | | А.А. Уткин |
| | <i>подпись</i> | | |

Воронеж 2020

Содержание

| | | |
|----------|---|-----------|
| 1 | Постановка задачи | 3 |
| 2 | Используемые инструменты | 4 |
| 3 | Общая структура программы | 5 |
| 4 | Общая структура библиотеки целых длинных чисел | 6 |
| 5 | Примеры работы библиотеки | 9 |
| 6 | Руководство пользователя | 12 |
| 7 | Блок-схема методов библиотеки | 13 |
| 8 | Исходный код программы | 43 |
| 8.1 | Исходный код bigint.py | 43 |
| 8.2 | Исходный код main.py | 52 |

1 Постановка задачи

1. Составить алгоритм (в виде блок-схемы) и написать (на любом языке программирования) соответствующую ему программу, позволяющую выполнять арифметические операции (сложение, вычитание, умножение и деление) над длинными целыми числами;
2. Составить алгоритм и написать соответствующую ему программу, позволяющую возводить целое число в квадрат;
3. Составить алгоритм и написать соответствующую ему программу, позволяющую возводить натуральное число в натуральную степень;
4. Составить алгоритм и написать соответствующую ему программу, позволяющую вычислить целую часть квадратного корня из натурального числа;
5. Составить алгоритм и написать соответствующую ему программу, позволяющую вычислить целую часть кубического корня из натурального числа;
6. Используя один из предложенных выше алгоритмов, составить блок-схему и написать соответствующую ей программу, позволяющую вычислять наибольший общий делитель двух больших натуральных чисел.

2 Используемые инструменты

Для решения вышеуказанных задач были использованы следующие инструменты:

- Основным ЯП был выбран Python версии 3.8.1;
- Для создания интерфейса был использован фреймворк Qt5, а также его расширение PyQt5;
- Для построение основы интерфейса была использована кроссплатформенная свободная среда для разработки графических интерфейсов программ использующих библиотеку Qt - Qt Designer;
- Для компиляции программы в бинарный файл .exe использован конвертер файлов Auto PY to EXE, который использует для своей работы PyInstaller.

3 Общая структура программы

Условно программу, написанную для решения вышеуказанных задач, можно разделить на две основных логических части:

1. Интерфейс пользователя.

Содержит в себе логику обработки команд, поступающих от пользователя. Содержит в себе код, отвечающий за разметку элементов интерфейса в окне, а также код, отвечающий за поведение программы, при использовании этих элементов;

2. Библиотека работы целых длинных чисел.

Содержит в себе обособленную часть кода, которая может быть подключена как отдельная библиотека к любой программе на ЯП Python.

4 Общая структура библиотеки целых длинных чисел

В библиотеке целых длинных содержится класс «BigInt», внутри которого находятся следующие методы:

- Сложение целых длинных чисел.

Программная реализация представляет собой сложение чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания. При этом данный способ не является медленно работающим;

- Вычитание целых длинных чисел.

Программная реализация представляет собой вычитание чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания. При этом данный способ не является медленно работающим;

- Умножение целых длинных чисел.

Программная реализация представляет собой умножение чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания. При этом данный способ не является медленно работающим;

- Целочисленное деление целых длинных чисел.

Программная реализация представляет собой деление чисел в «столбик» без дробной части. Данный способ реализации был выбран по причине простоты его работы и написания. При этом данный способ не является медленно работающим;

- Выделение корня из простого длинного числа любой положительной целой степени.

Программная реализация представляет подбор наиболее близкого числа, возведенного в данную из аргументов степень, при котором результат возведения в степень не будет превышать число, из которого выделяется корень. Выбор данного способа обусловлен простотой его

реализации, а также отсутствием предполагаемых альтернатив. При этом, скорее всего, альтернативы есть;

- Возведение в степень простого длинного числа.

Программная реализация представляет умножение данного числа на самого себя, используя рекурсивные вызовы этой же функции. Размер этого повторного умножения равно числу, в степень которого необходимо возвести некоторое число. Выбор данного способа реализации обусловлено желанием опробовать рекурсию на практике.

Класс «BigInt» содержит в себе два основных поля:

1. Поле хранения числа «value».

Представляет собой переменную типа строка, в которой содержится число экземпляра класса;

2. Поле хранения знака числа «is_neg».

Представляет собой переменную типа bool, в которой содержится информация о знаке числа. Значение True эквивалентно отрицательному числу, значение False - положительному;

Создания экземпляра класса «BigInt» происходит следующими способами:

- Создание экземпляра класса без передачи аргументов. Числовое значение такого экземпляра будет равно нулю.

```
1 a = BigInt()  
2
```

- Создание экземпляра класса с передачей в аргумент строки, которая может валидно быть приведена к типу целого числа.

```
1 a = BigInt('-1234567890') # a = -1234567890  
2 b = BigInt('1234567890') # b = 1234567890  
3 d = BigInt('0')          # d = 0  
4
```

- Создание экземпляра класса с передачей в аргумент целого числа.

```
1 a = BigInt(-1234567890) # a = -1234567890
2 b = BigInt(1234567890)  # b = 1234567890
3 d = BigInt(0)           # d = 0
4
```

- Создание экземпляра класса с передачей в аргумент экземпляра класса «BigInt».

```
1 a = BigInt(-1234567890) # a = -1234567890
2 b = BigInt(a)           # b = -1234567890
3
```

Также в данной библиотеке содержится функция «GCD», реализующая возможность нахождения наибольшего общего делителя. Эта функция может работать как с экземплярами класса «BigInt», так и с численными типами данных ЯП Python.

5 Примеры работы библиотеки

В качестве примера работы будут использоваться прямые вызовы методов класса «BigInt». При этом, при работе с графической программой результаты будут идентичны.

Пусть даны два целых длинных числа a и b , сохраненных в экземпляр класса «BigInt». А так же, создадим экземпляр класса «BigInt» с нулевым значением.

```
1 a = BigInt('-1234567890987654321')
2 b = BigInt('9876543210123456789')
3 zero = BigInt()
4
```

- Выполним сложение:

```
1 print(a + b)
2
```

Вывод: 8641975319135802468

- Выполним вычитание:

```
1 print(a - b)
2
```

Вывод: -11111111101111111110

- Выполним умножение:

```
1 print(a * b)
2
```

Вывод: -12193263121170553265523548251112635269

- Выполним целочисленное деление:

```
1 print(a / b)
2
```

Вывод: -8

- Выполним нахождение остатка от деления:

```
1 print(b % a)
```

```
2
```

Вывод: 8222222221

- Выполним нахождение НОД:

```
1 print(GCD(b, a))
```

```
2
```

Вывод: -9

- Выполняем возведение в степень:

```
1 print(a.bipow(20))
```

```
2
```

Вывод:

67654945781131788253399139476950939867213847384221510782372183
38736383554932818216005379411615896402318839463975841663187950
47266740645217094738013218419327830527872057771151857381511749
91352856101226216668855950857925749095871686783571452199421977
46524667992025003348862025953101533163689052346013027443912327
3028907724064631250587670777171351261244651206462401

- Выполним деление на ноль:

```
1 print(a / zero)
```

```
2
```

Вывод: *ZeroDivisionError*

- Выполним деление нуля:

```
1 print(zero / b)
```

```
2
```

Вывод: 0

- Выполним умножение на ноль:

```
1 print(a * zero)
```

```
2
```

Вывод: 0

- Выполняем возведение в степень ноль:

```
1 print(b.bipow(0))
```

```
2
```

Вывод: 1

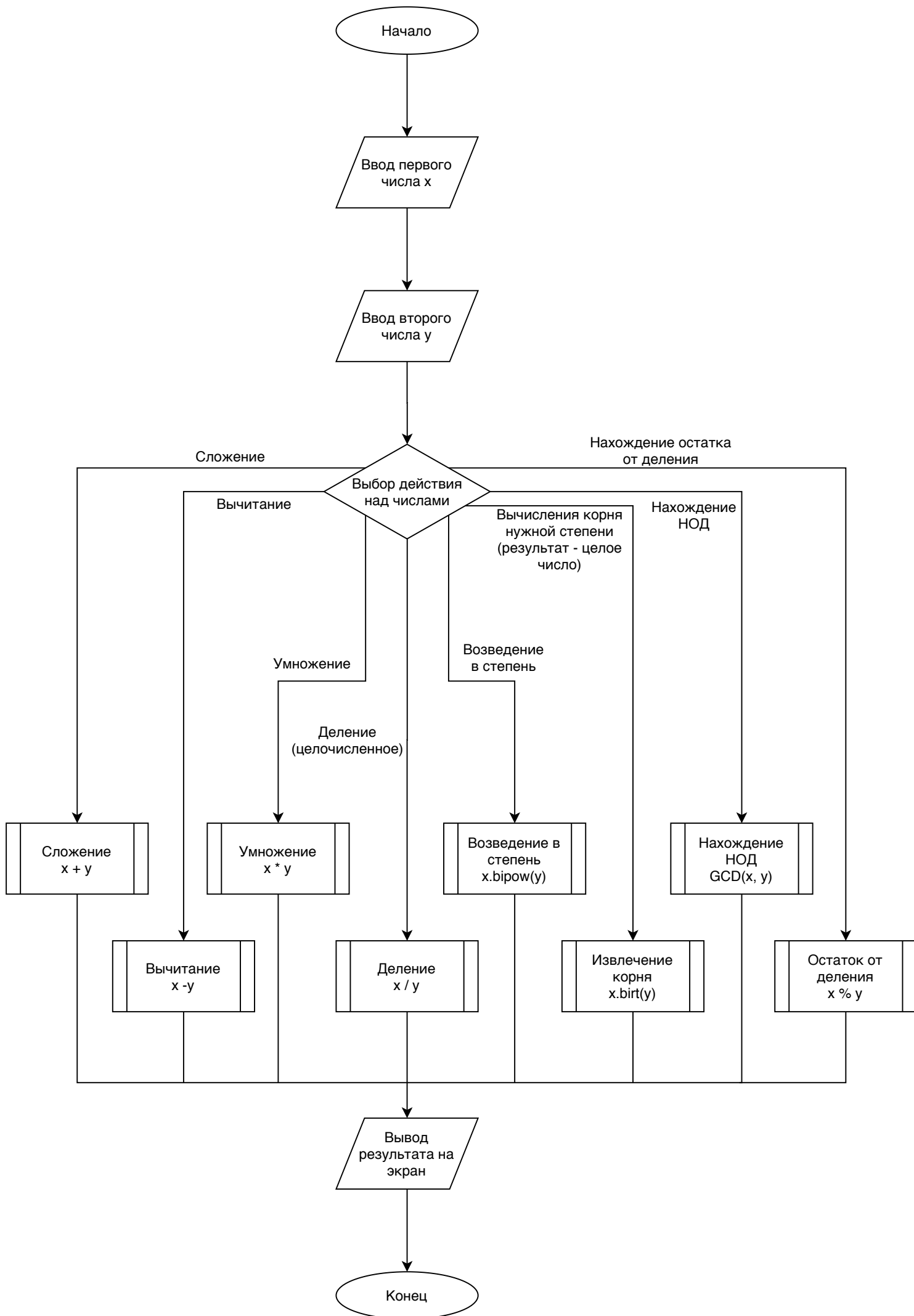
6 Руководство пользователя

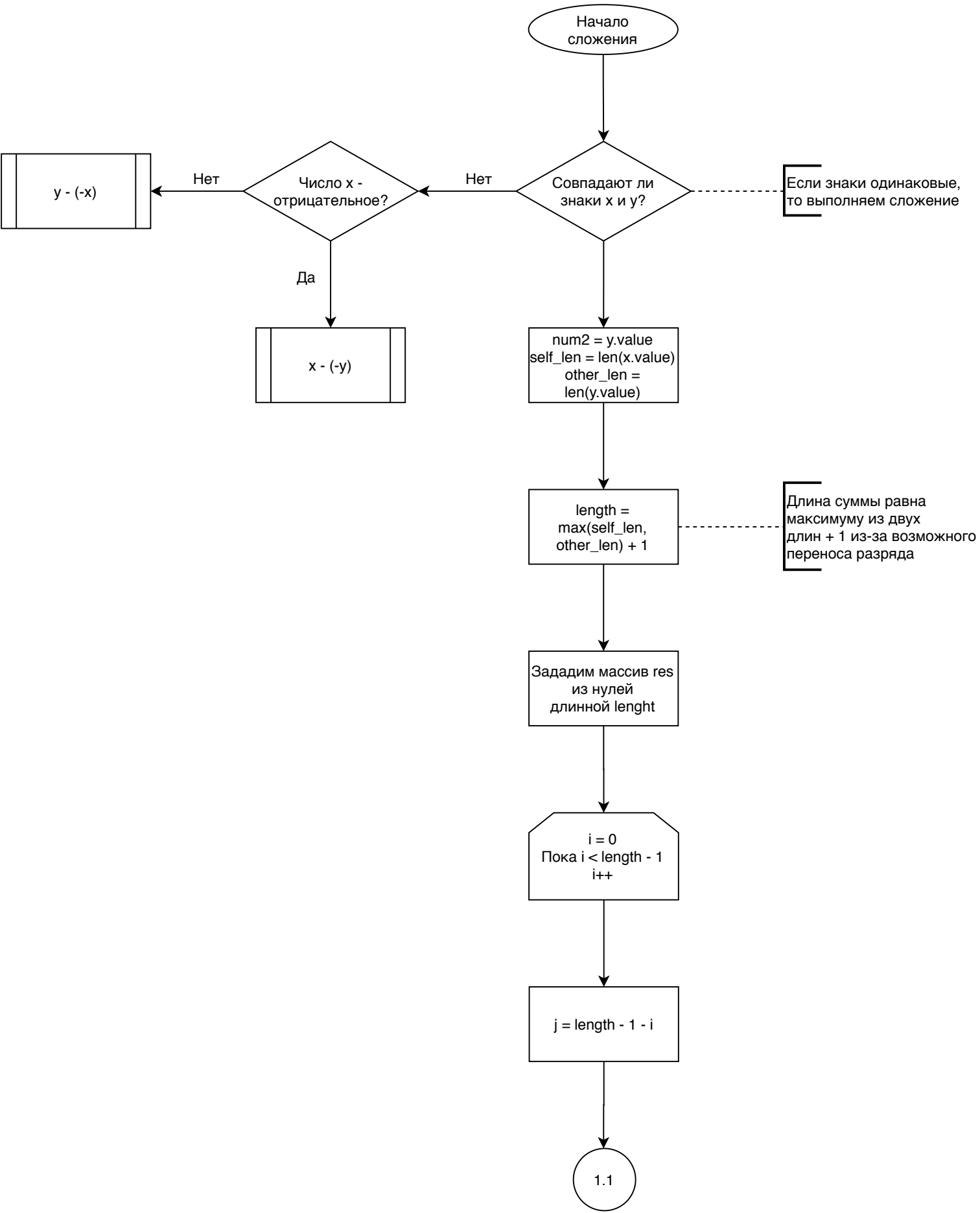
1. В случае с сложением, вычитанием, умножением и делением программа работает по принципу: [первое число] [действие] [второе число]
2. В случае возведения в степень программа работает по принципу: [первое число] в степени [второе число]
3. В случае извлечения корня ($\sqrt{\quad}$) программа работает по принципу: корень в степени [второе число] по [первое число]
4. В случае нахождения НОД программа ищет наибольший общий делитель чисел.
5. В случае нахождения НОД программа ищет остаток от деления первого числа на второе число.

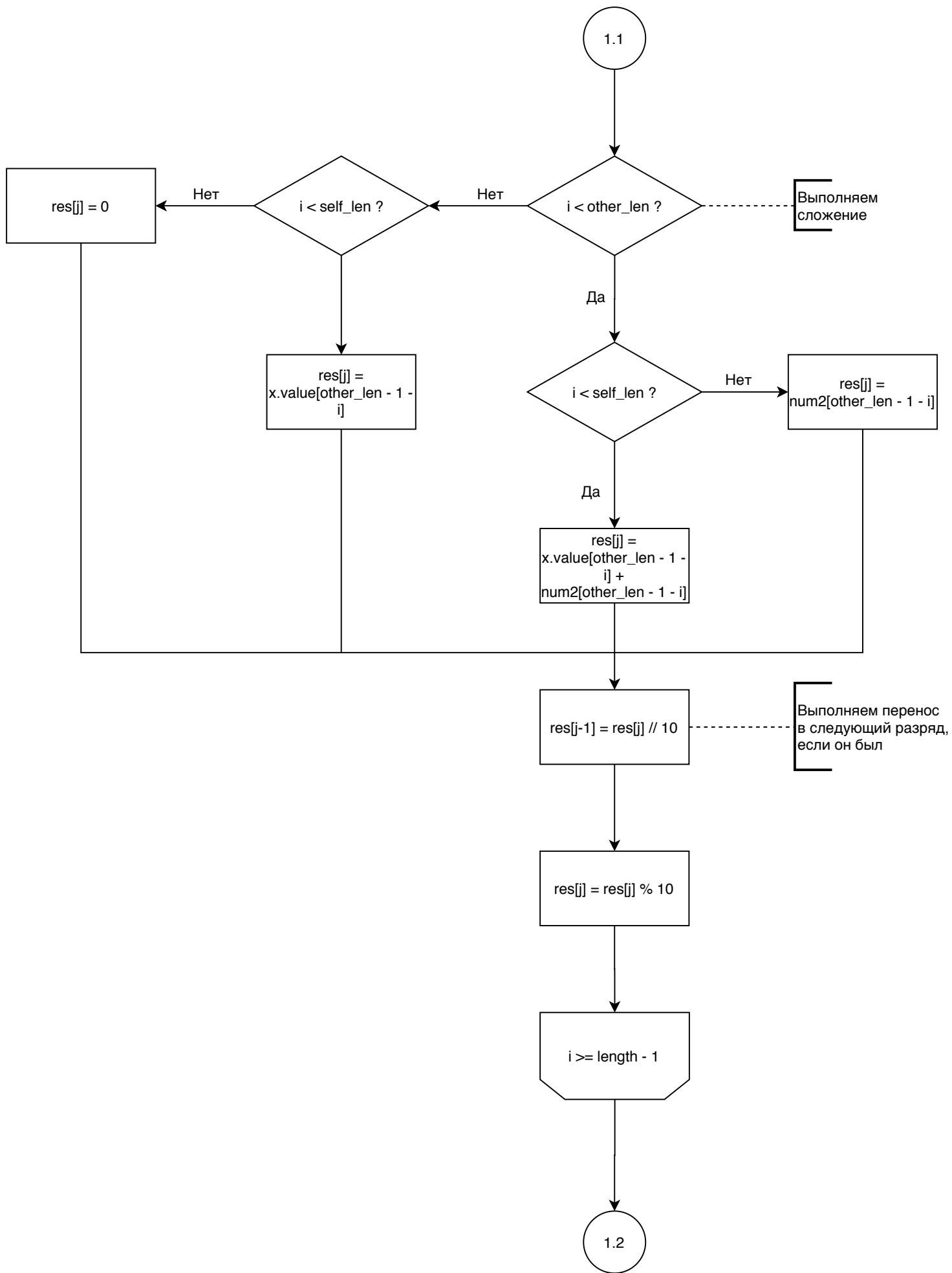
7 Блок-схема методов библиотеки

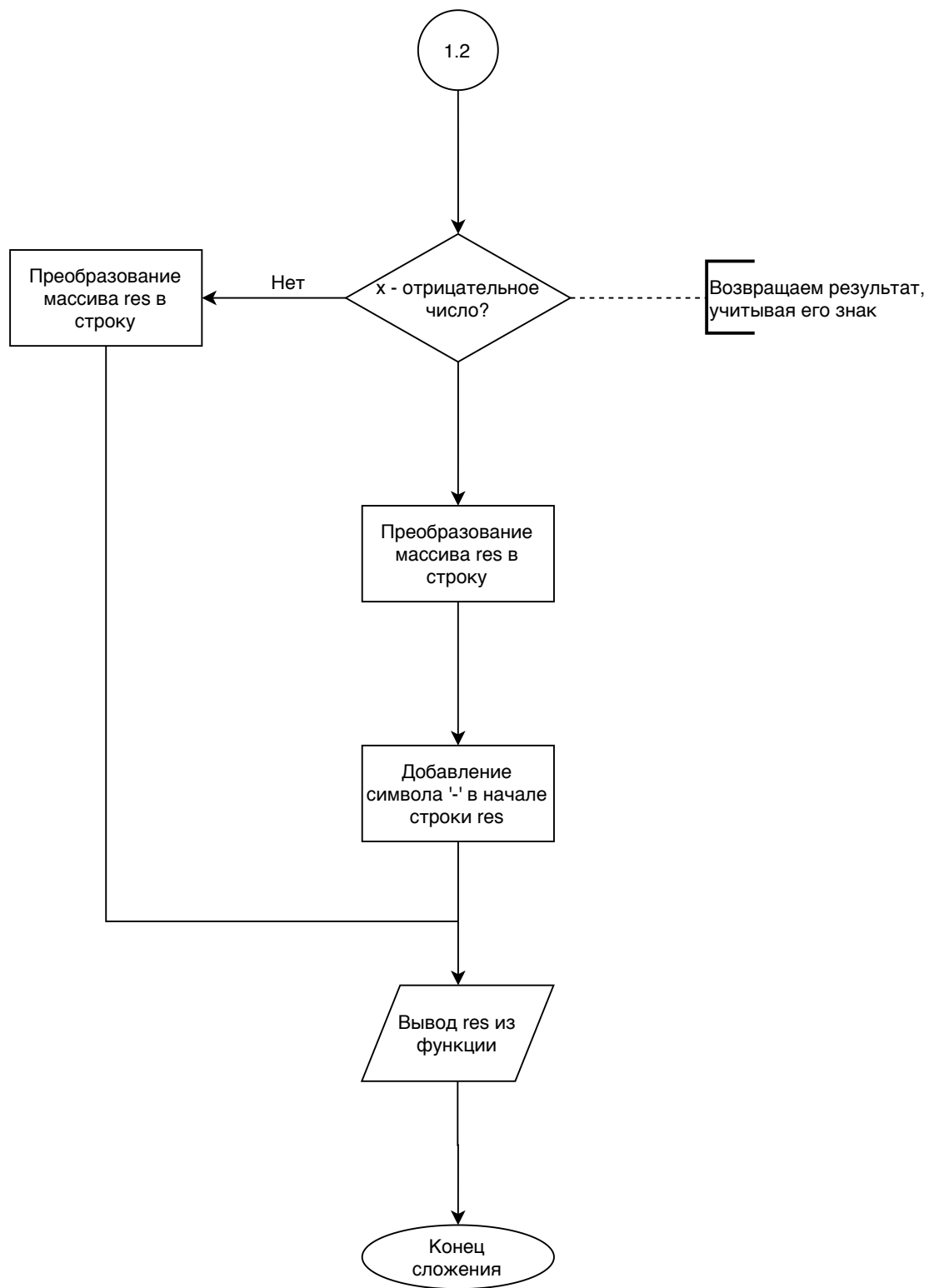
Ниже представлены блок-схемы методов в следующем порядке:

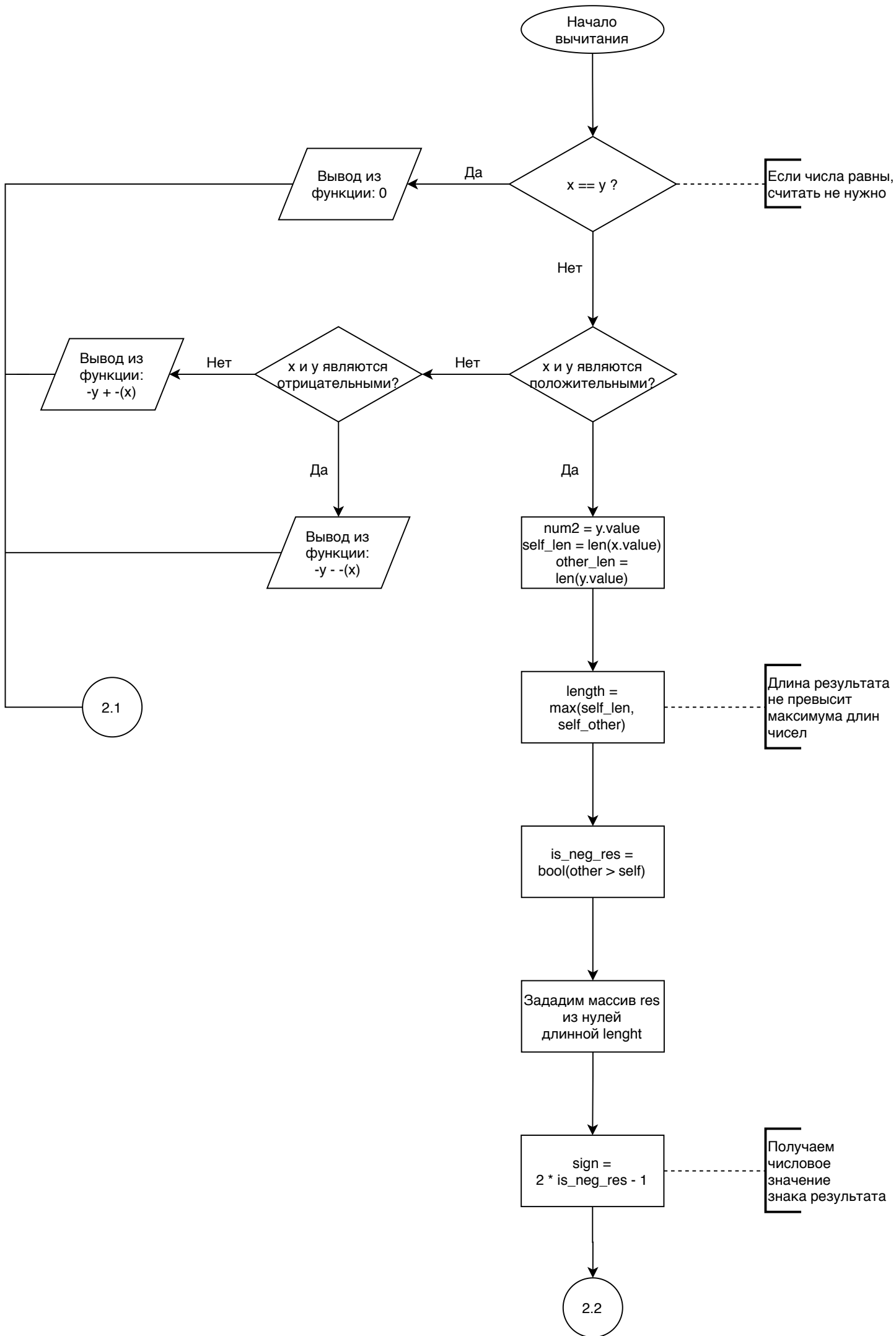
1. Логика работы интерфейса;
2. Сложение;
3. Вычитание;
4. Умножение;
5. Деление;
6. Возведение в степень;
7. Извлечение корня;
8. Нахождение НОД;
9. Нахождение остатка от деления.

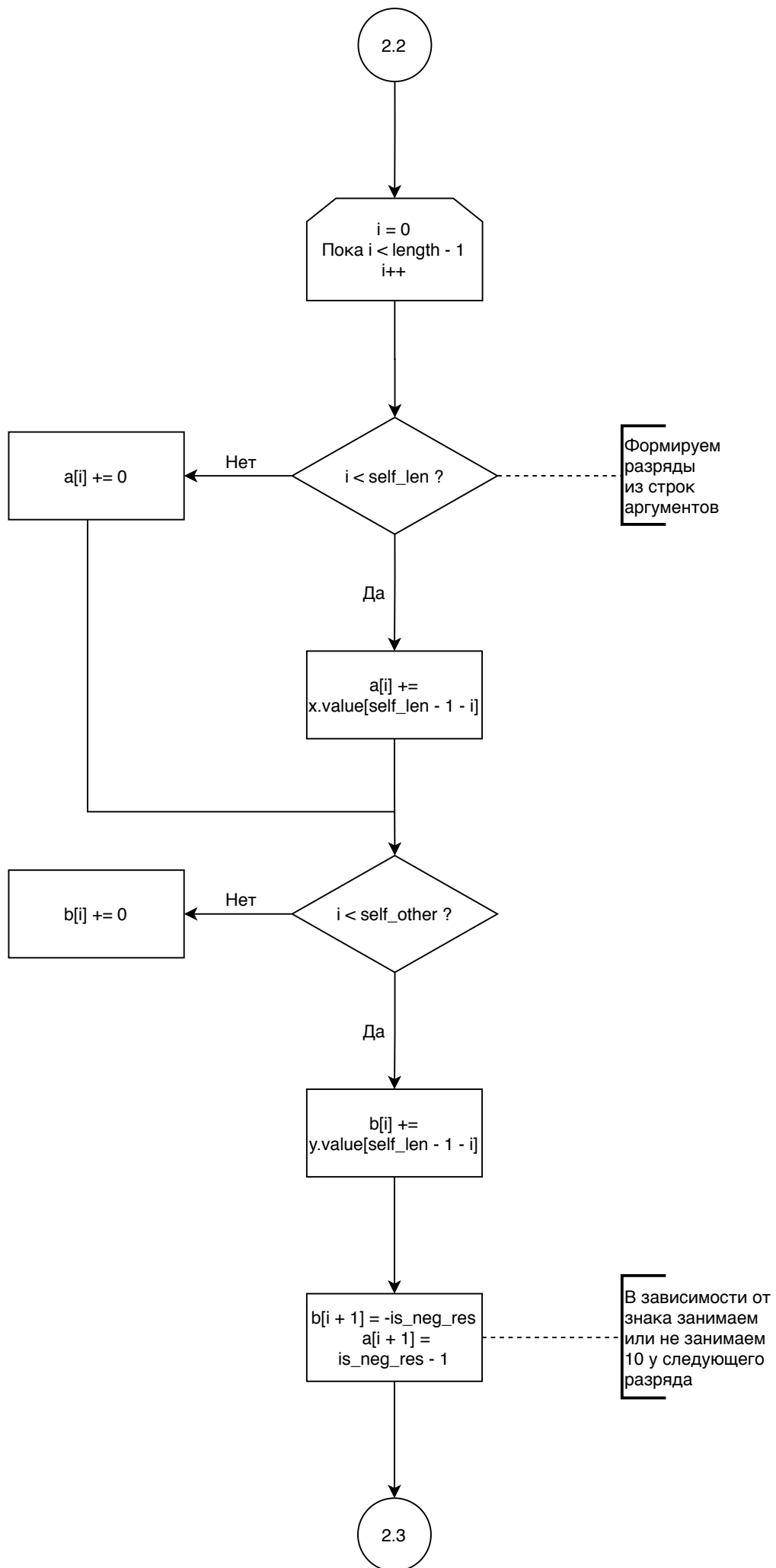


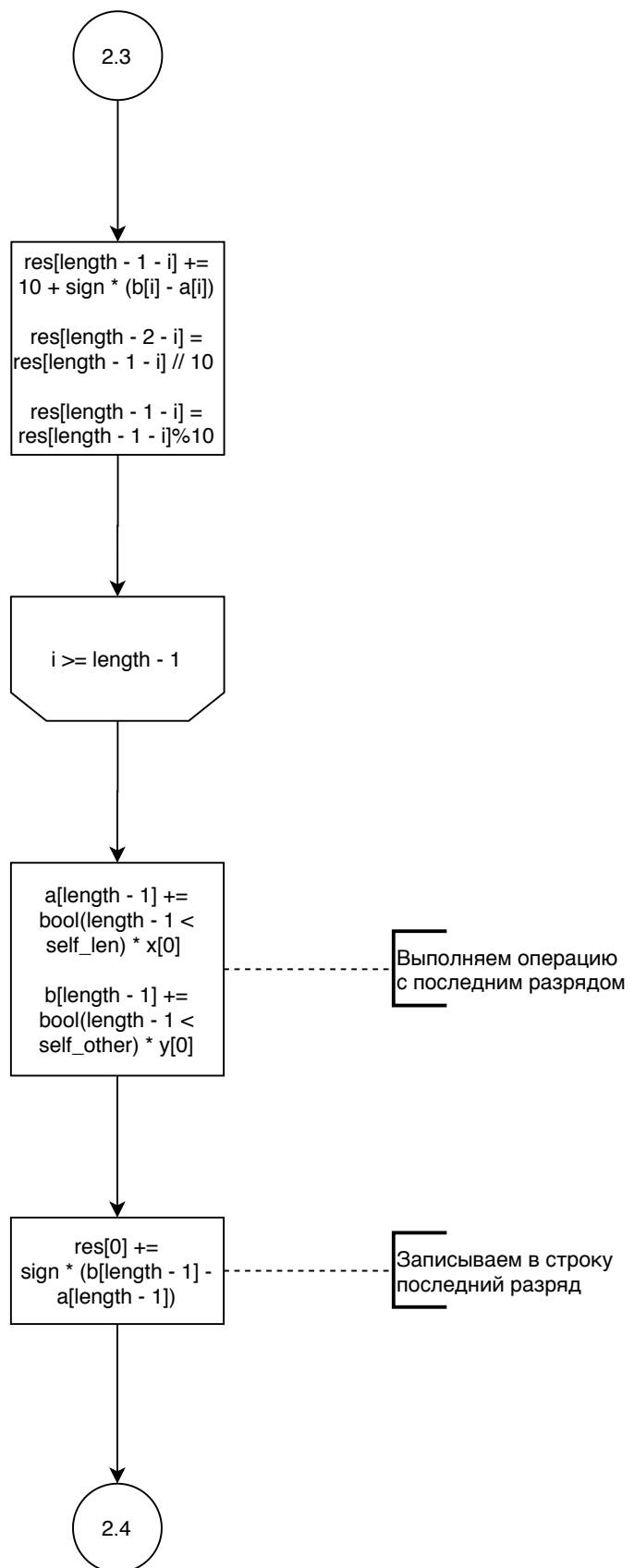


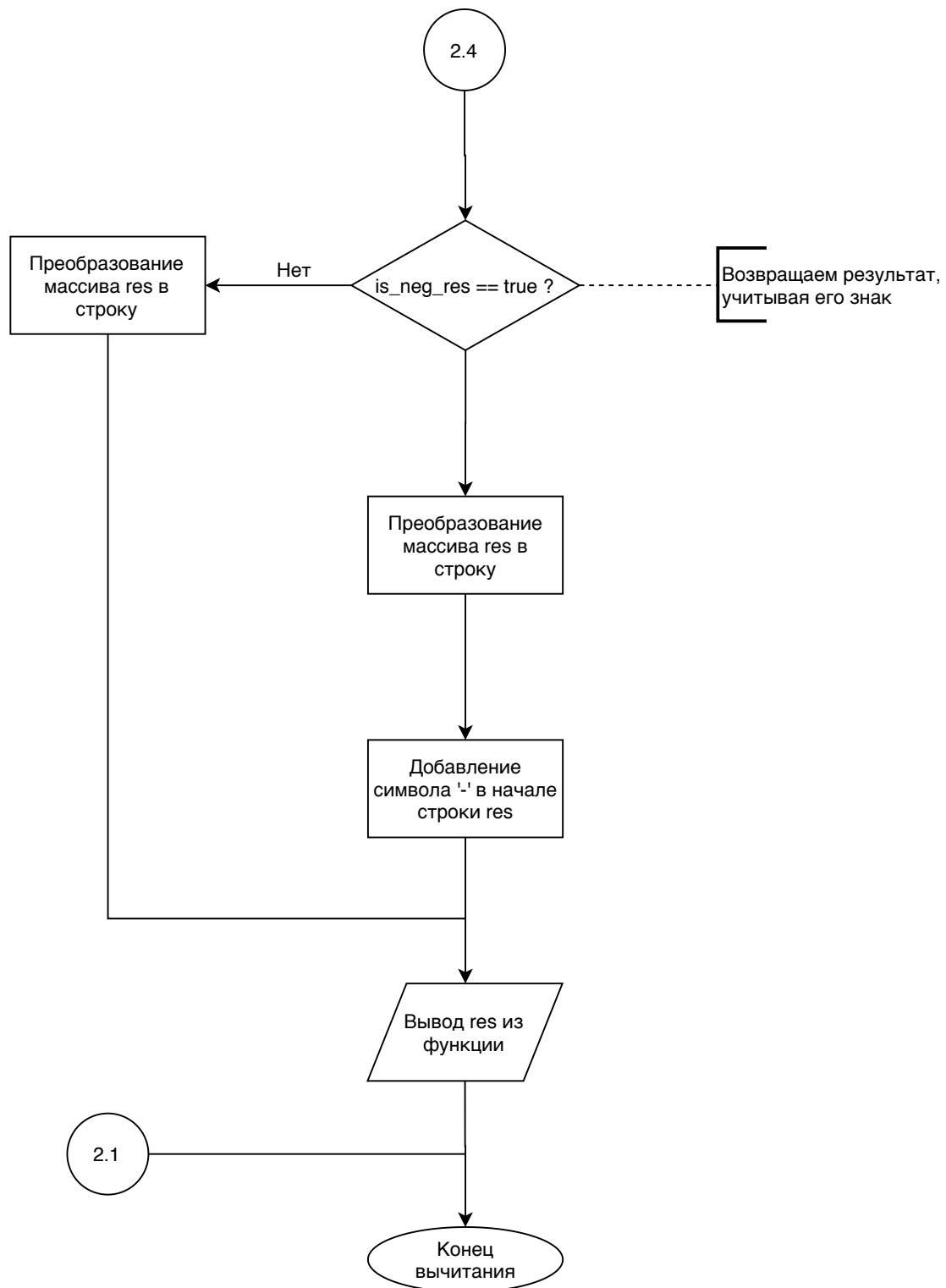


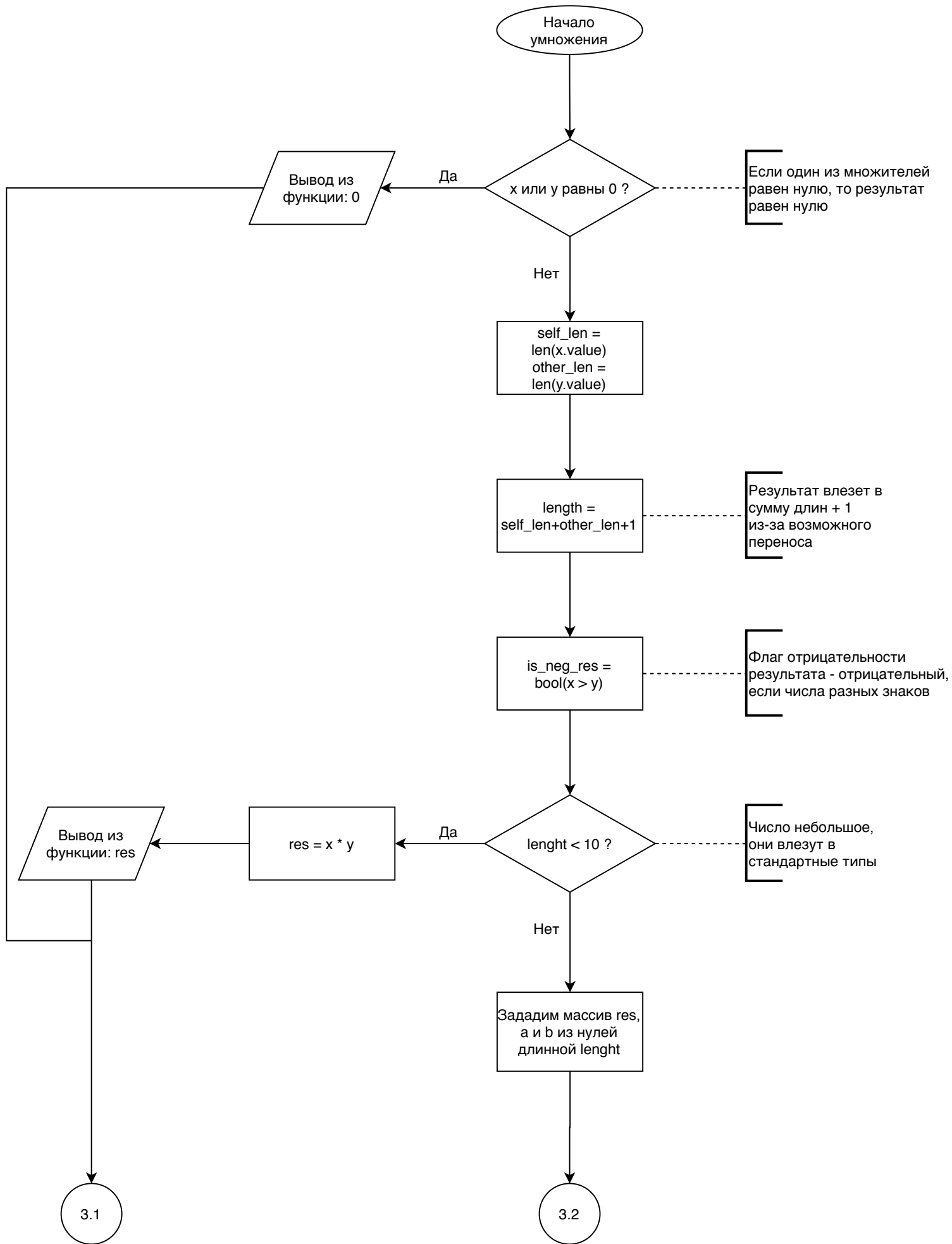


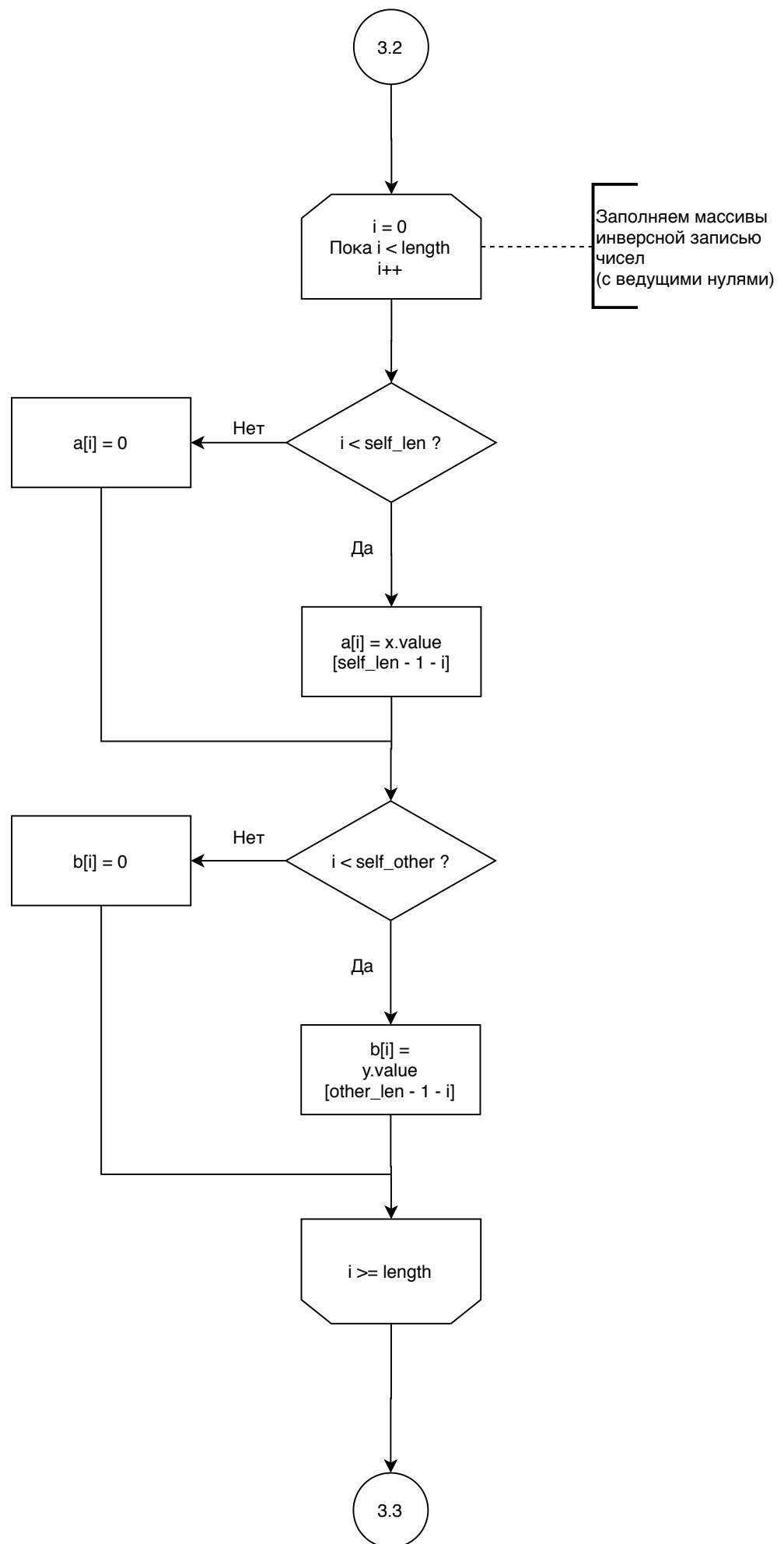


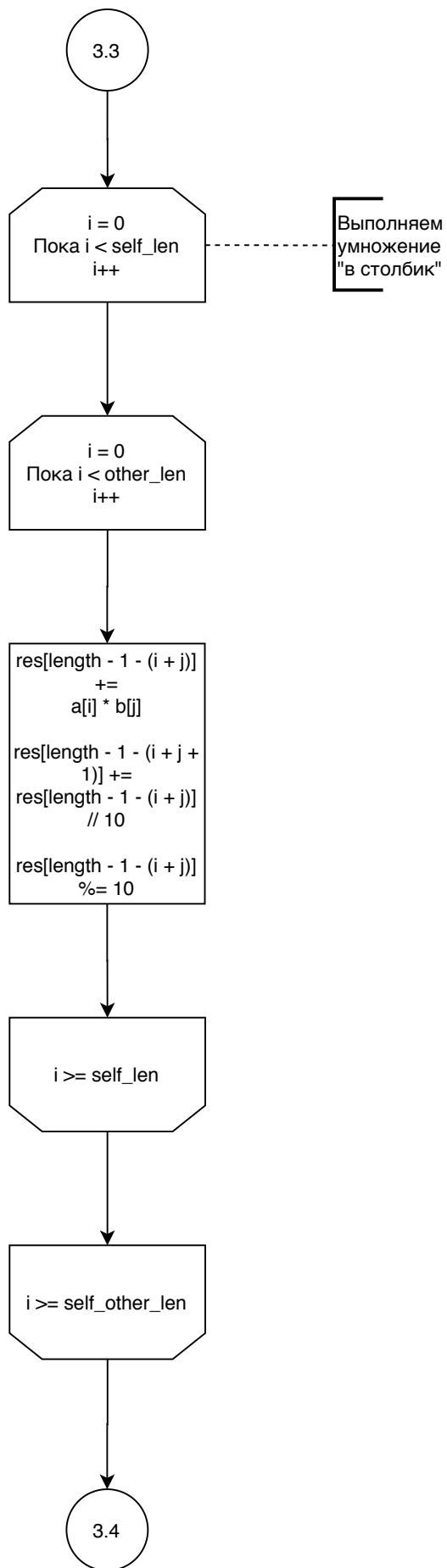


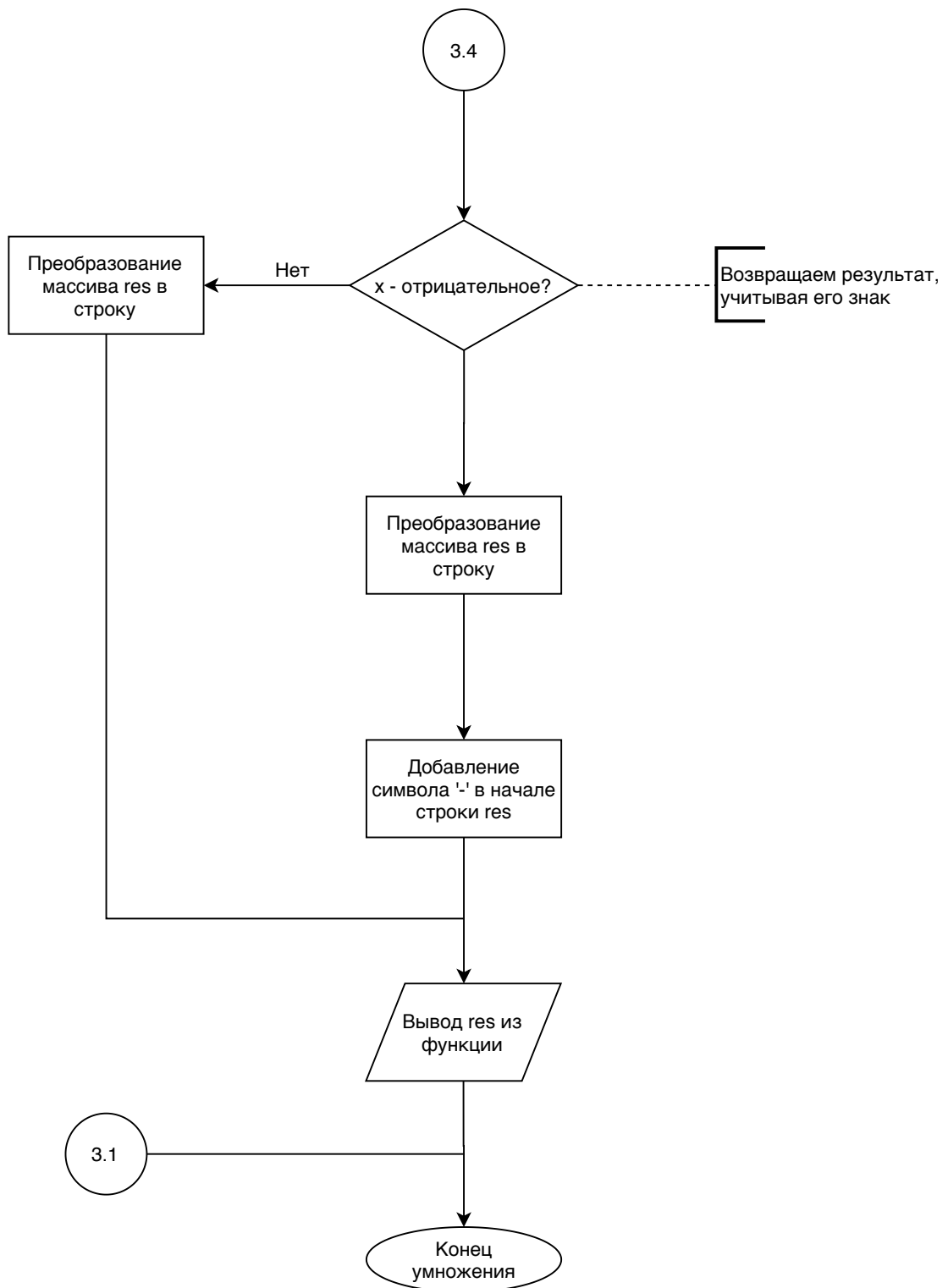


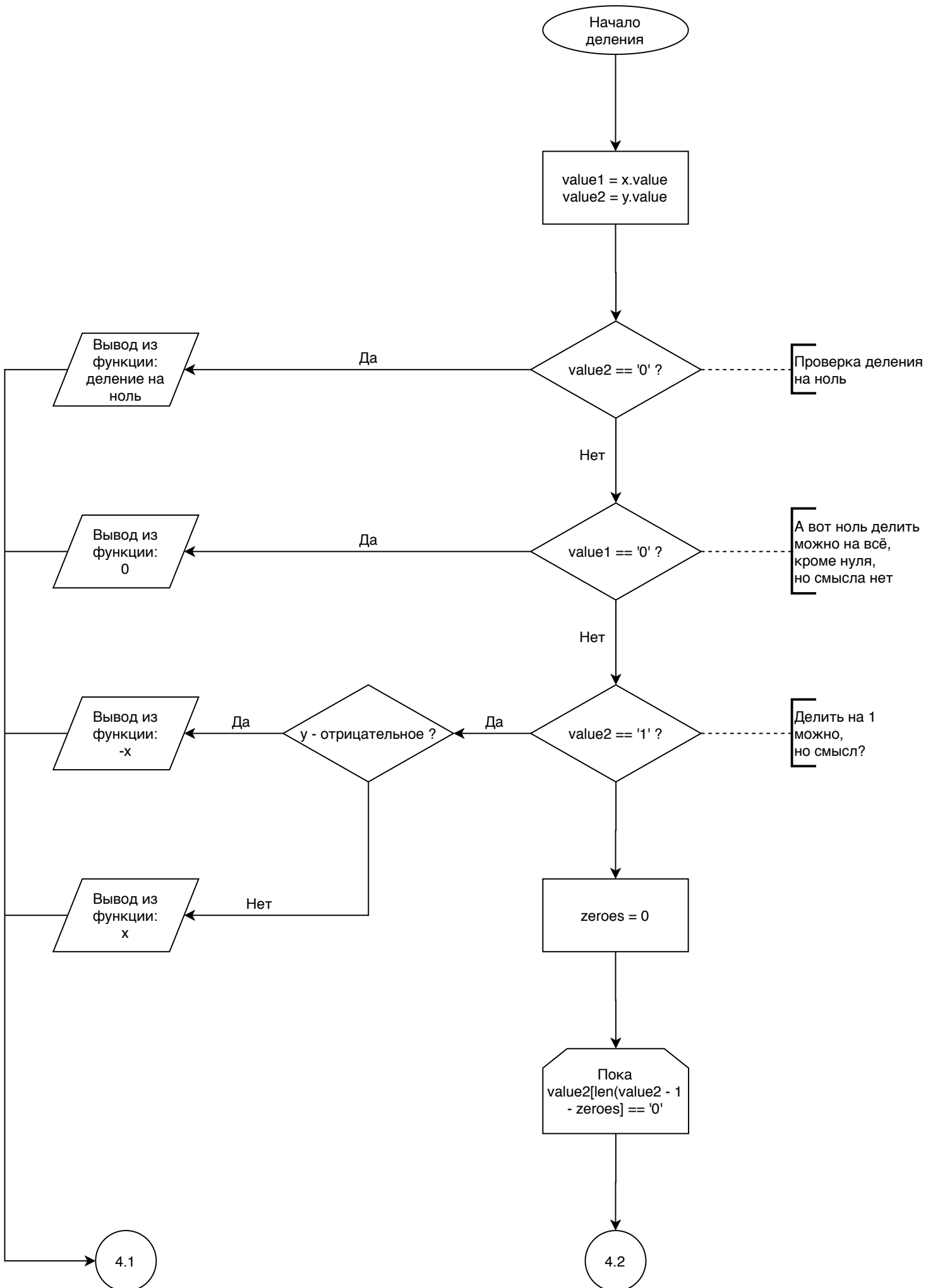


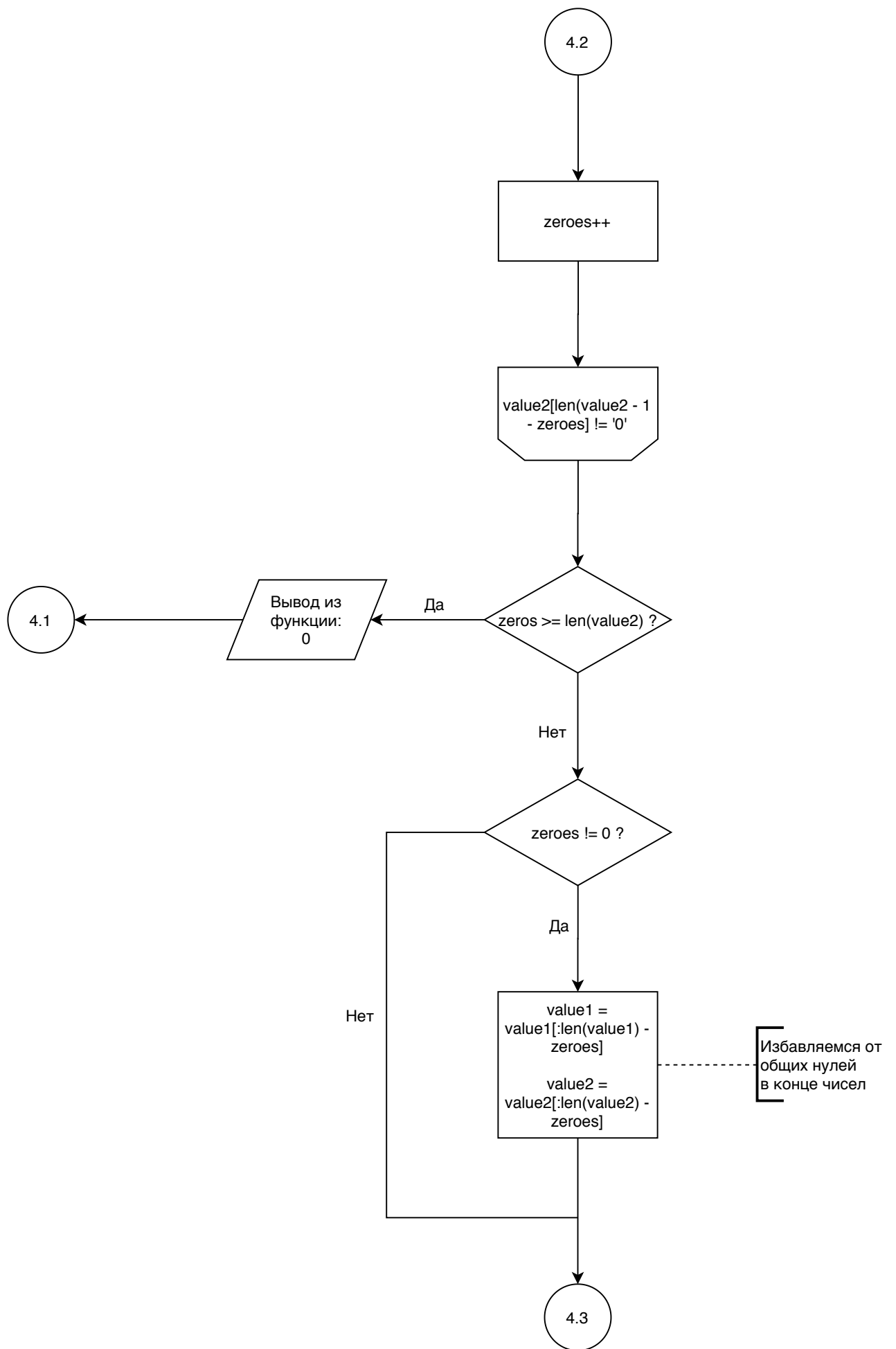


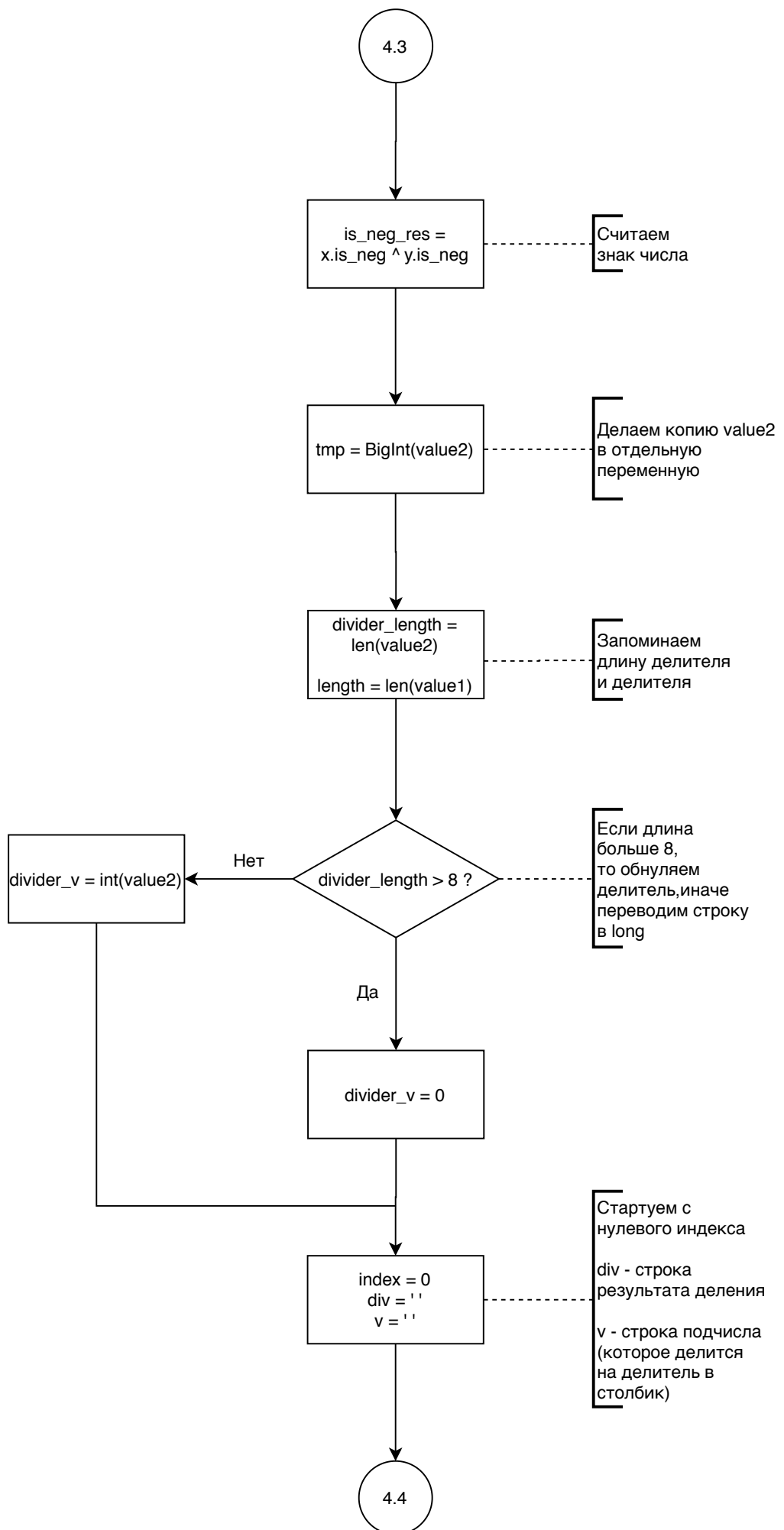


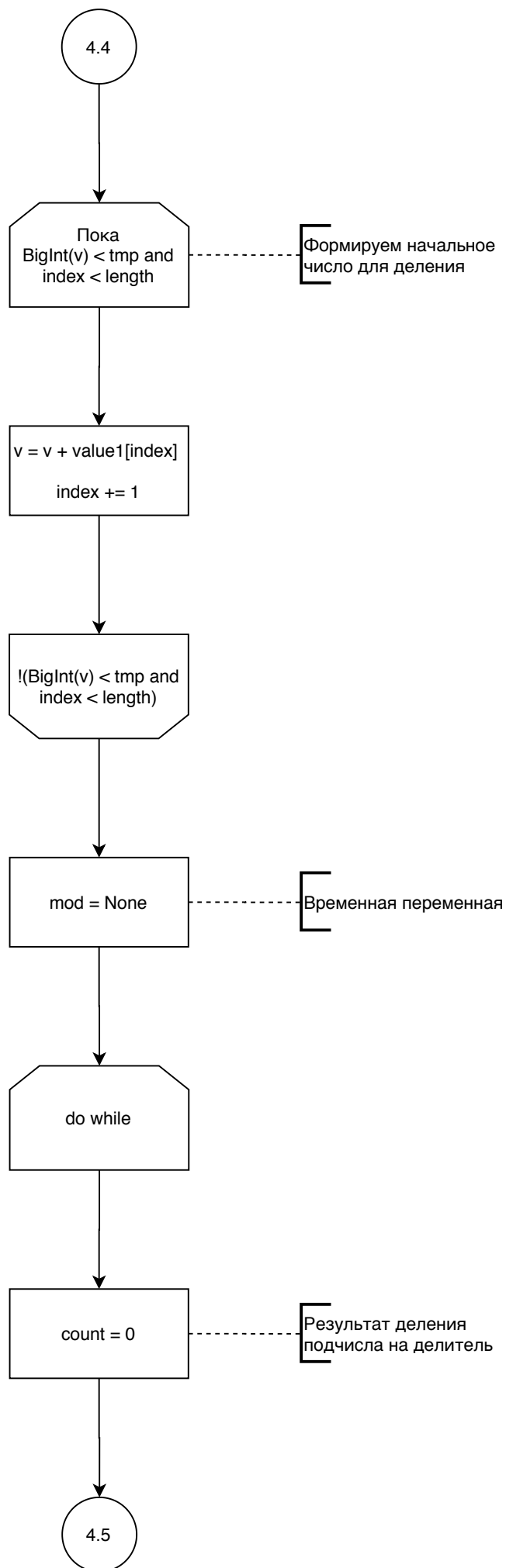


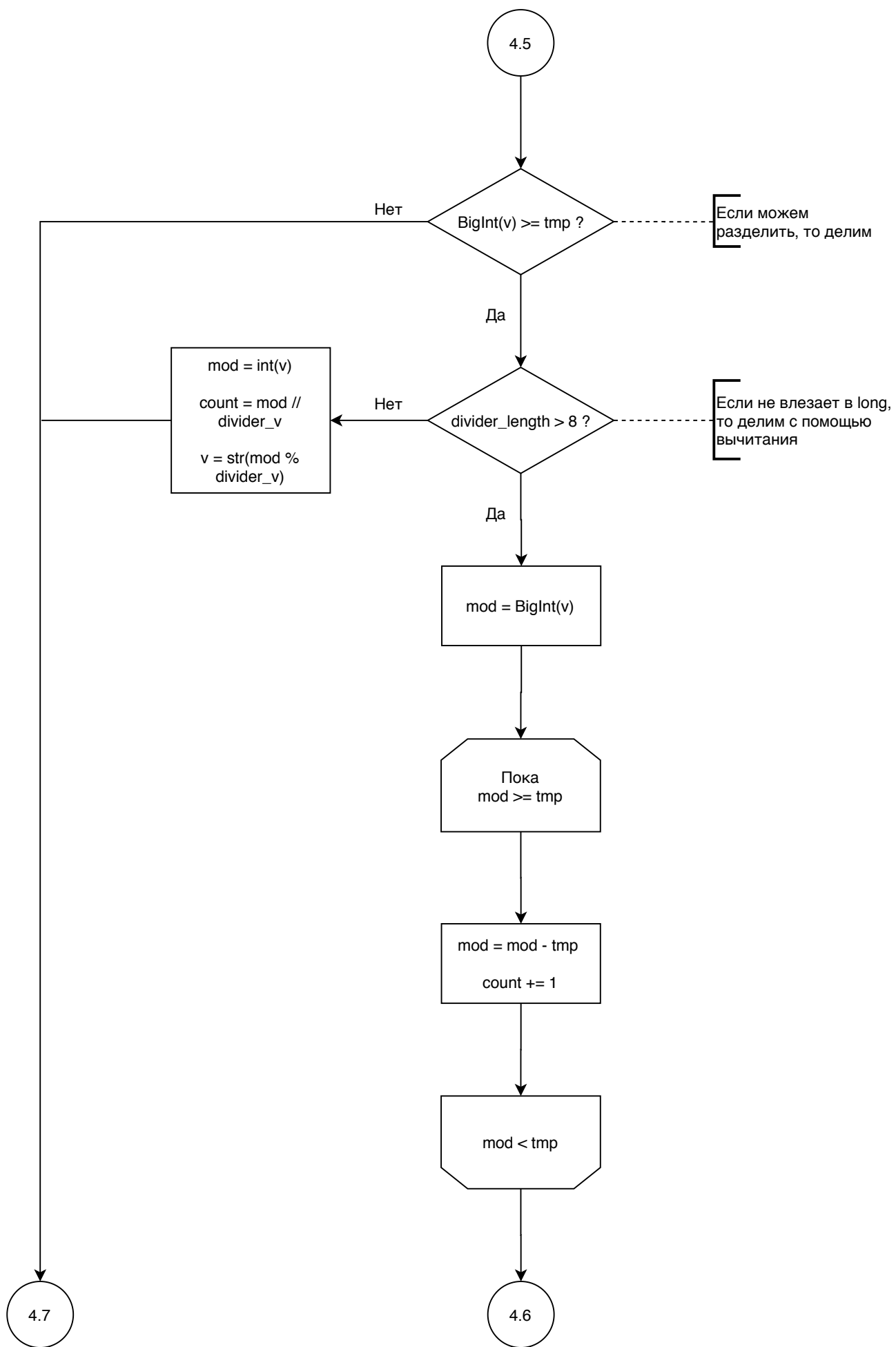


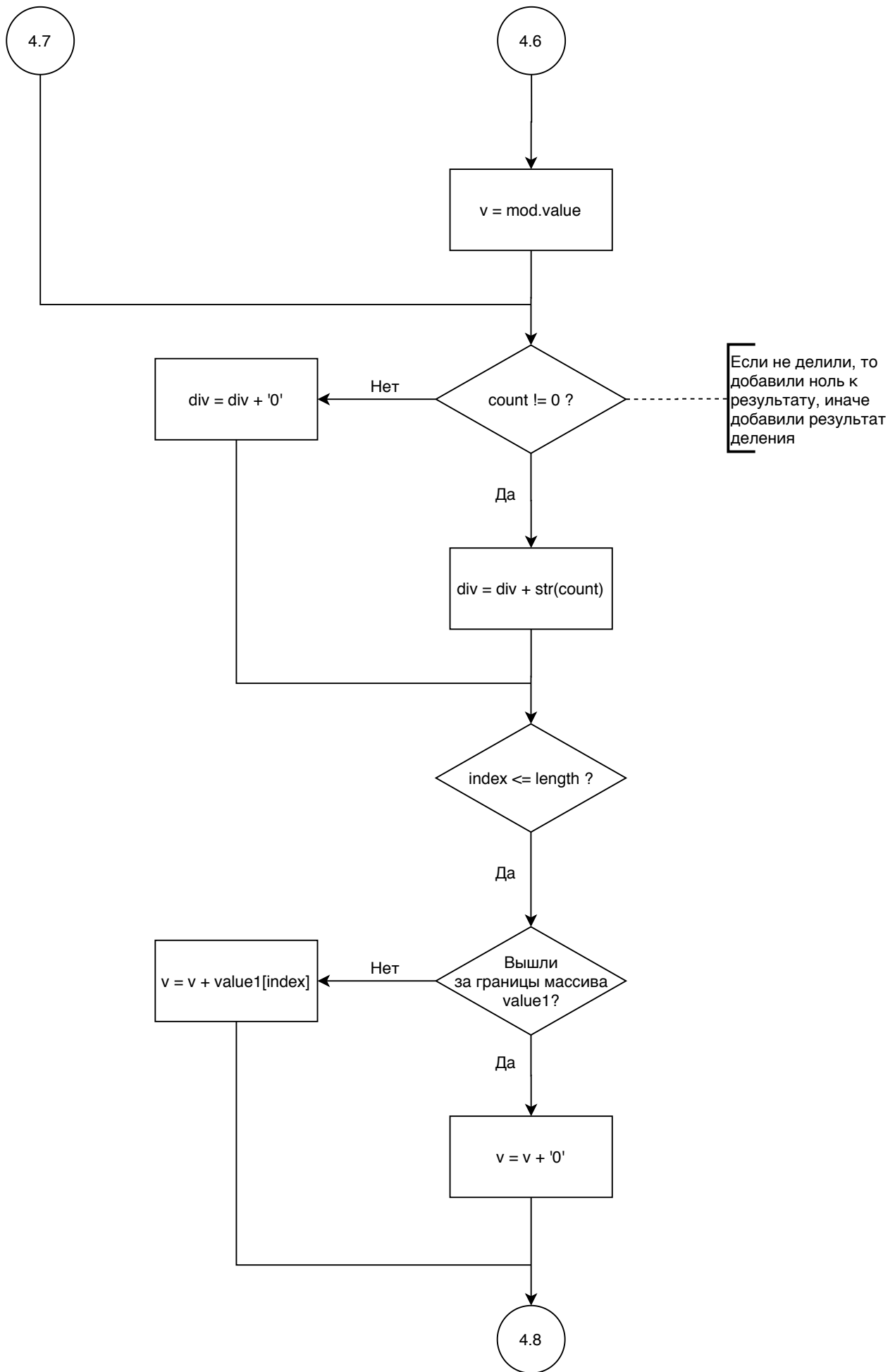


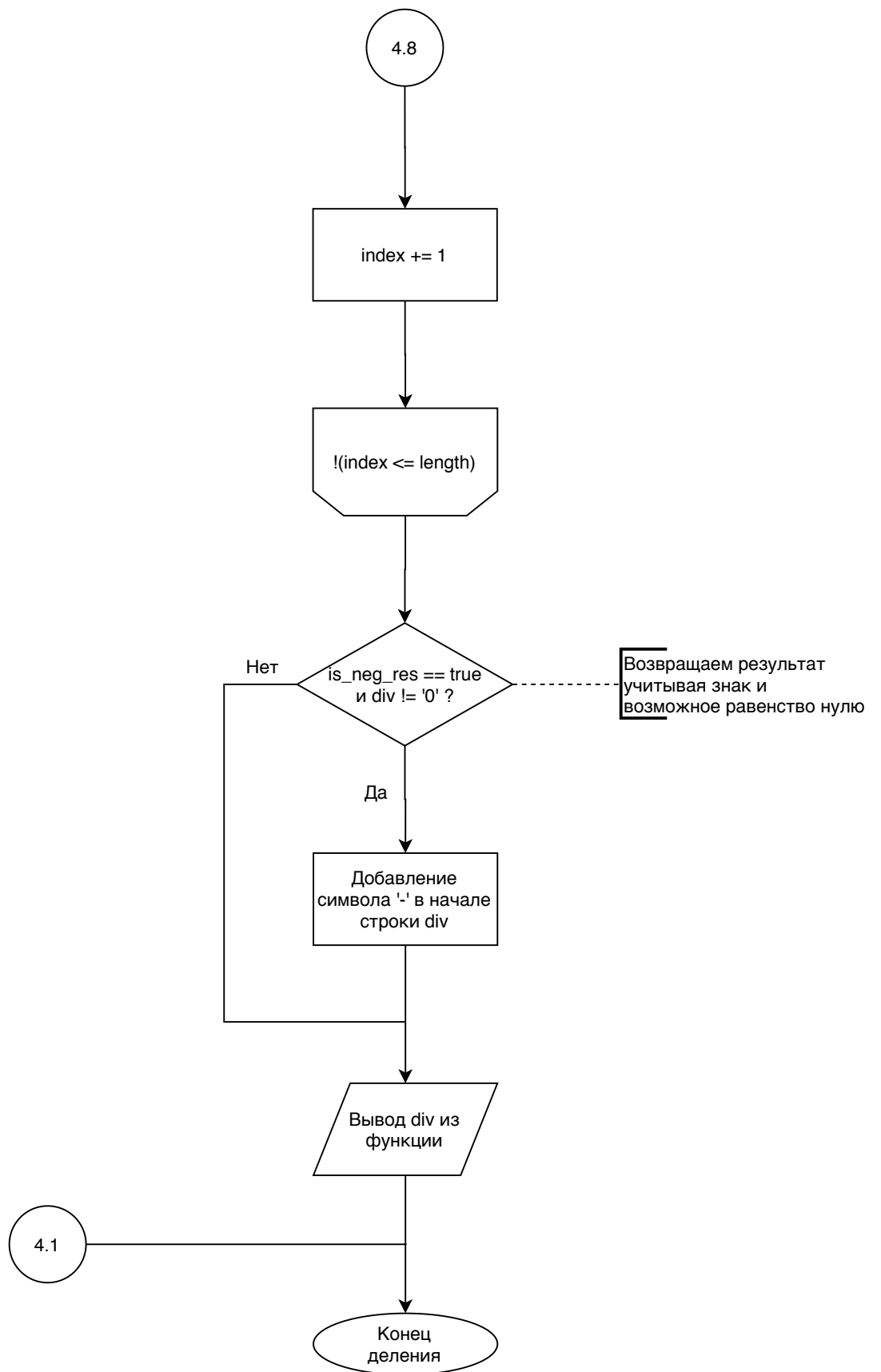


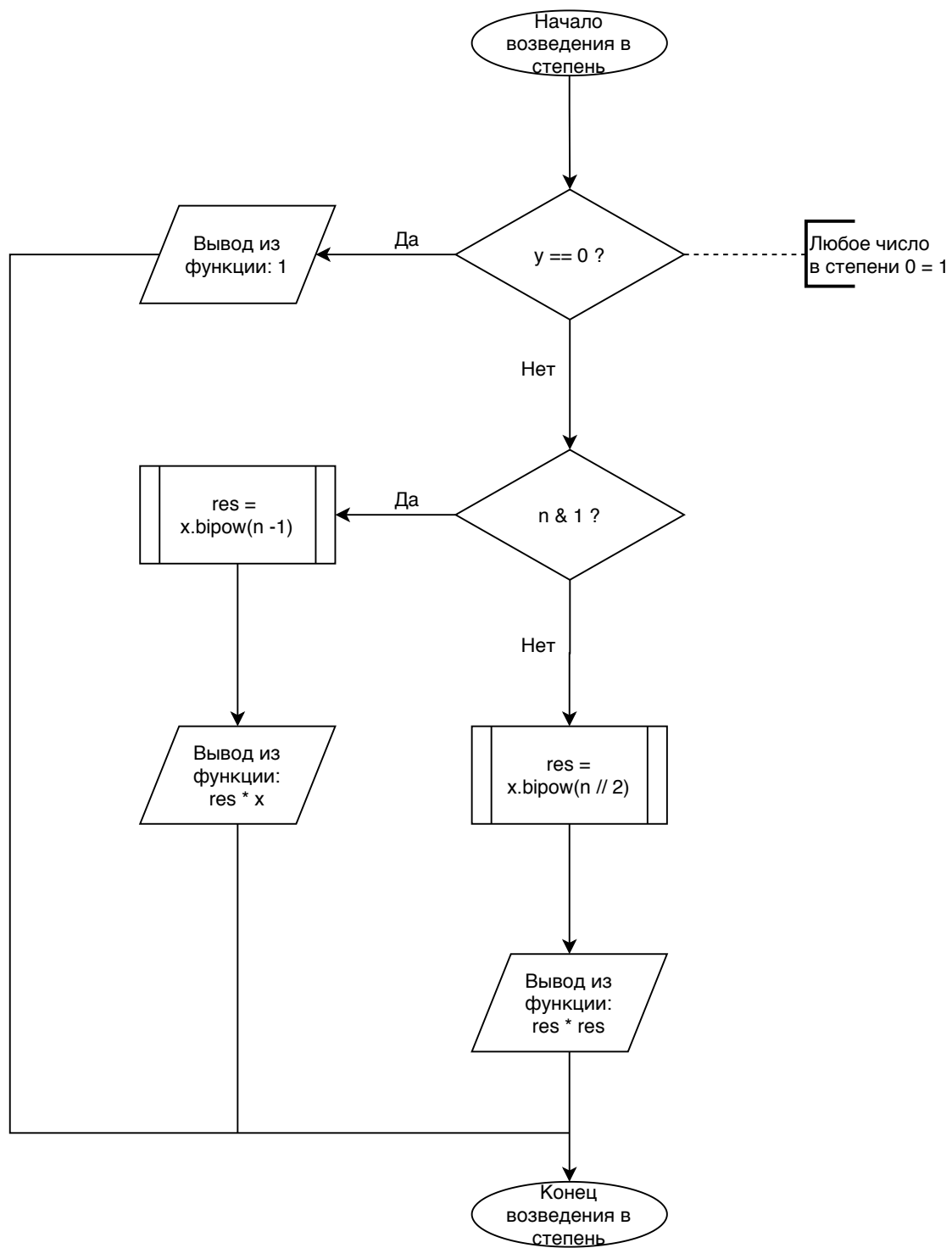


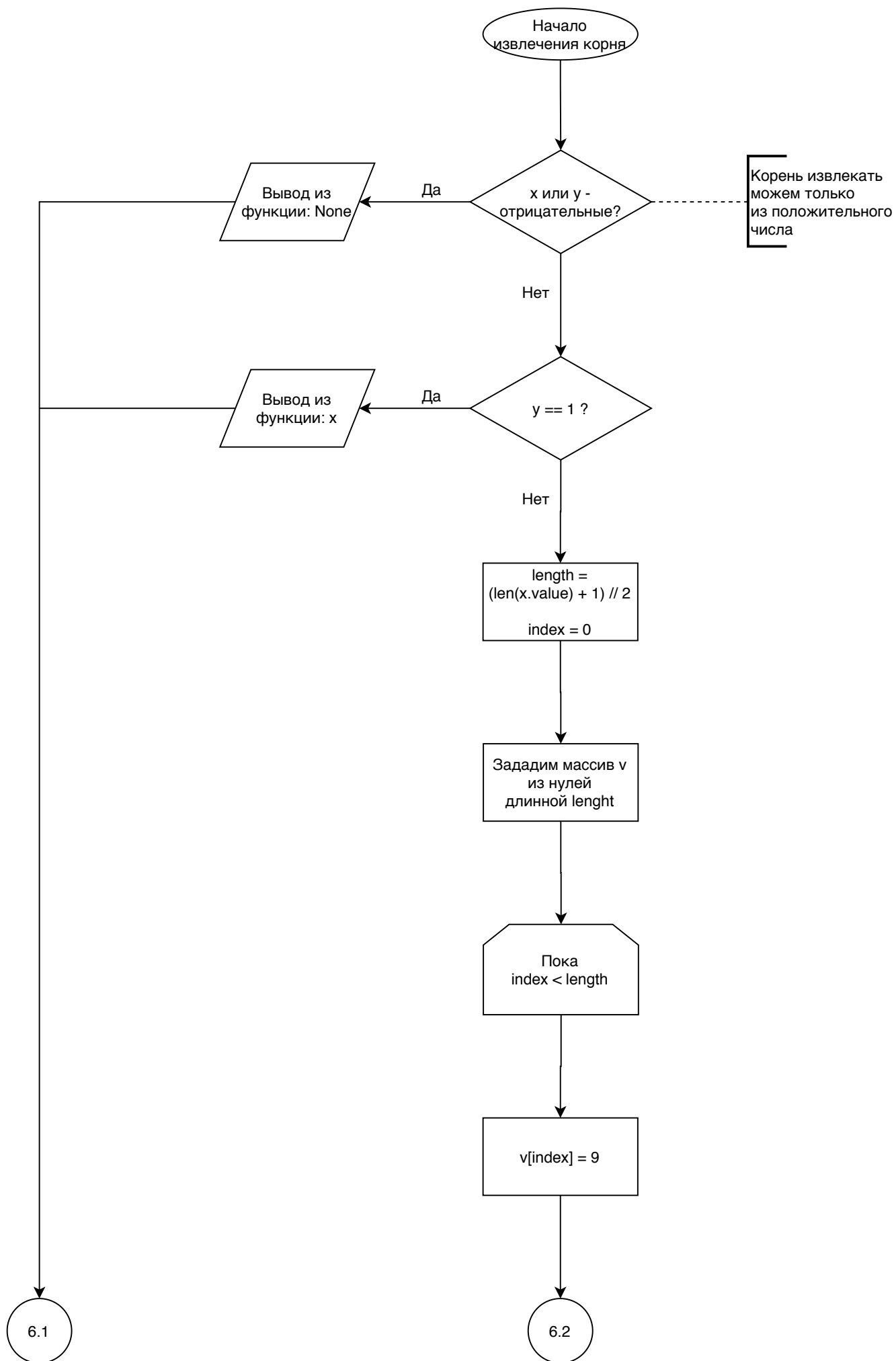




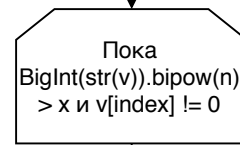




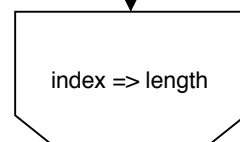
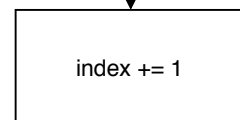
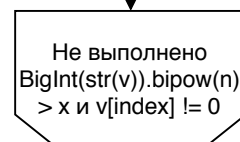
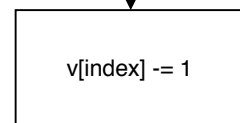




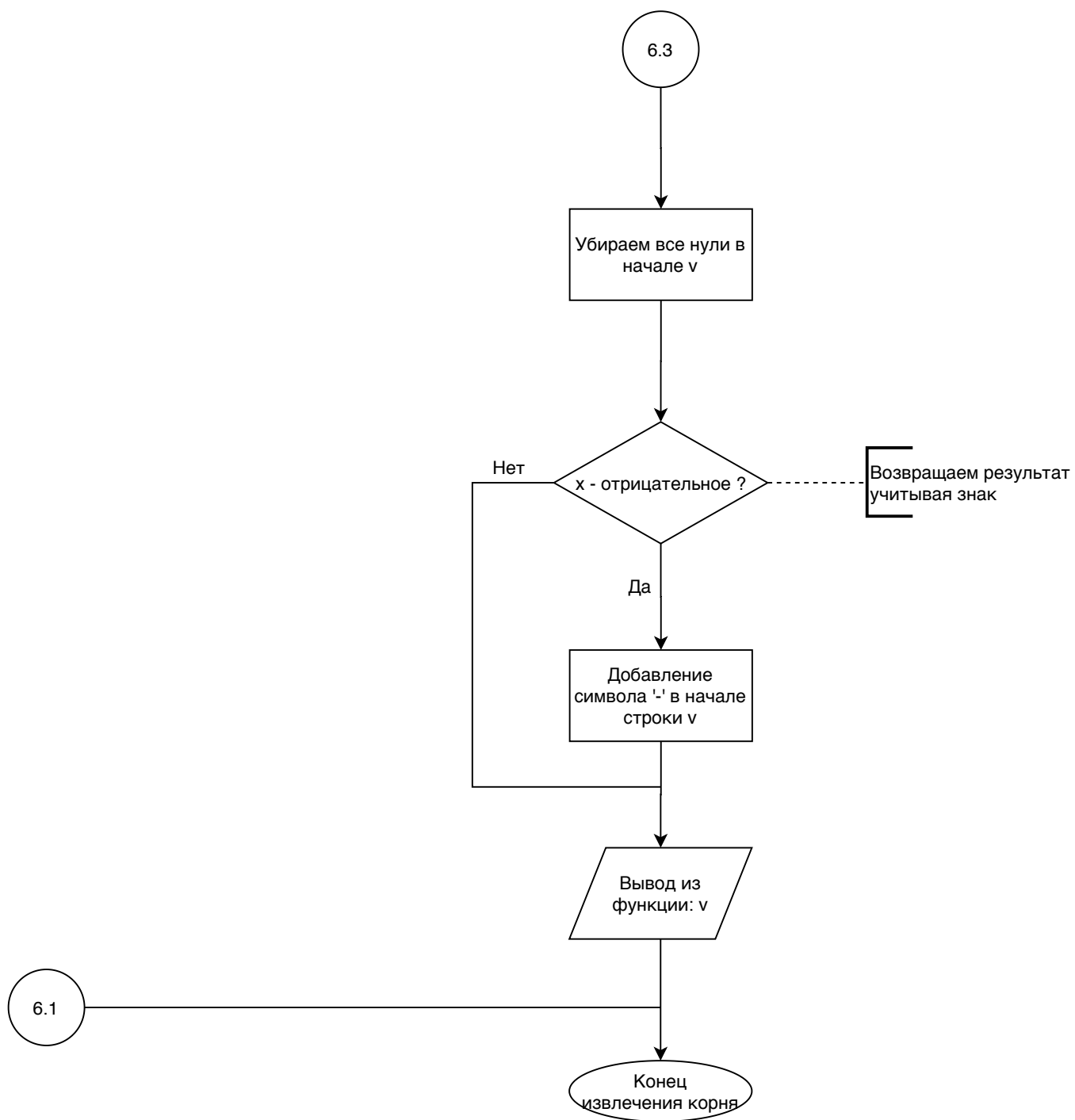
6.2

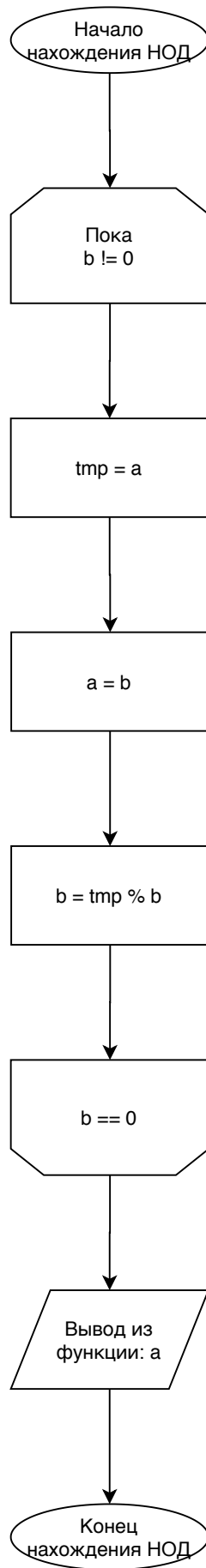


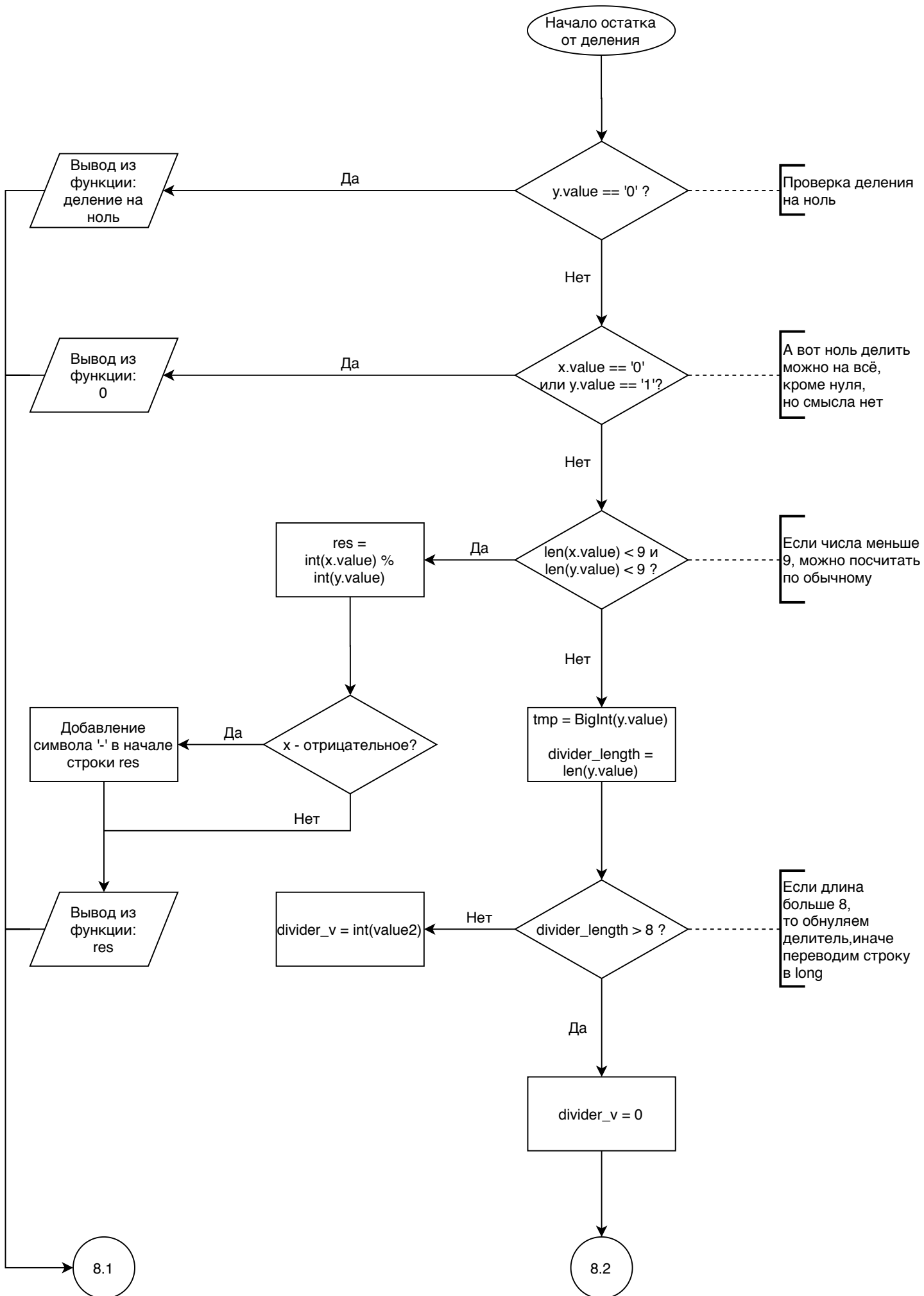
Начало подбора наиболее
близкого числа в степени n,
которое будет больше x

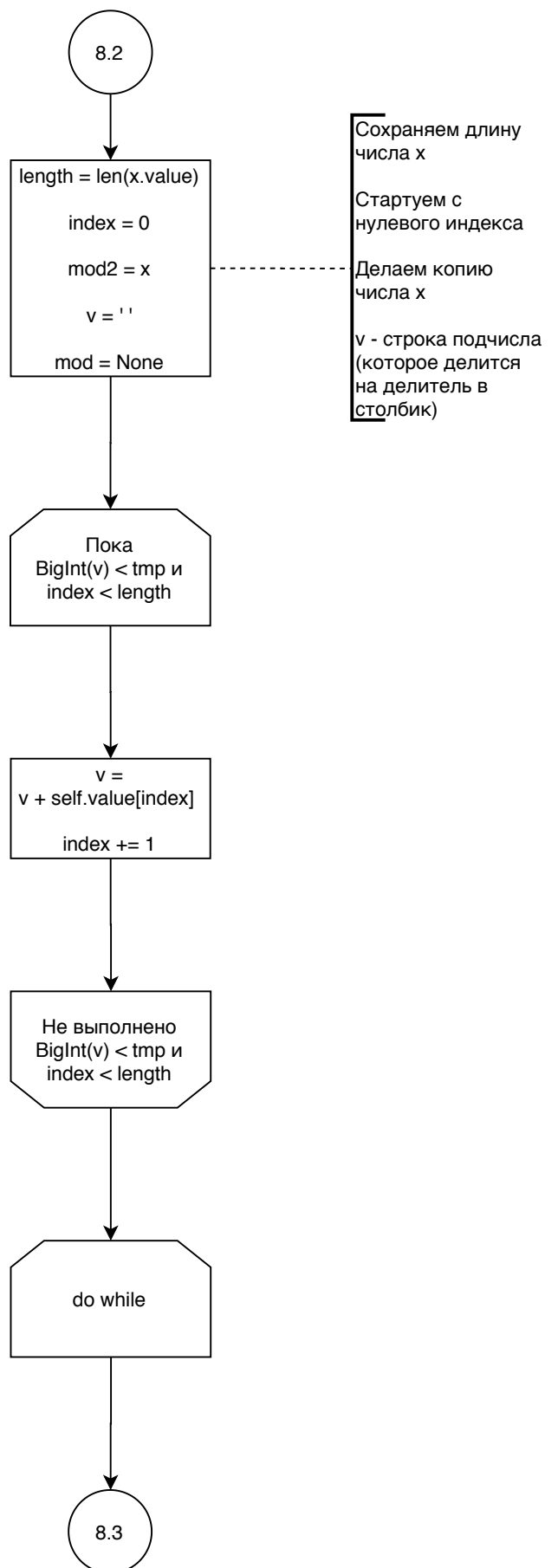


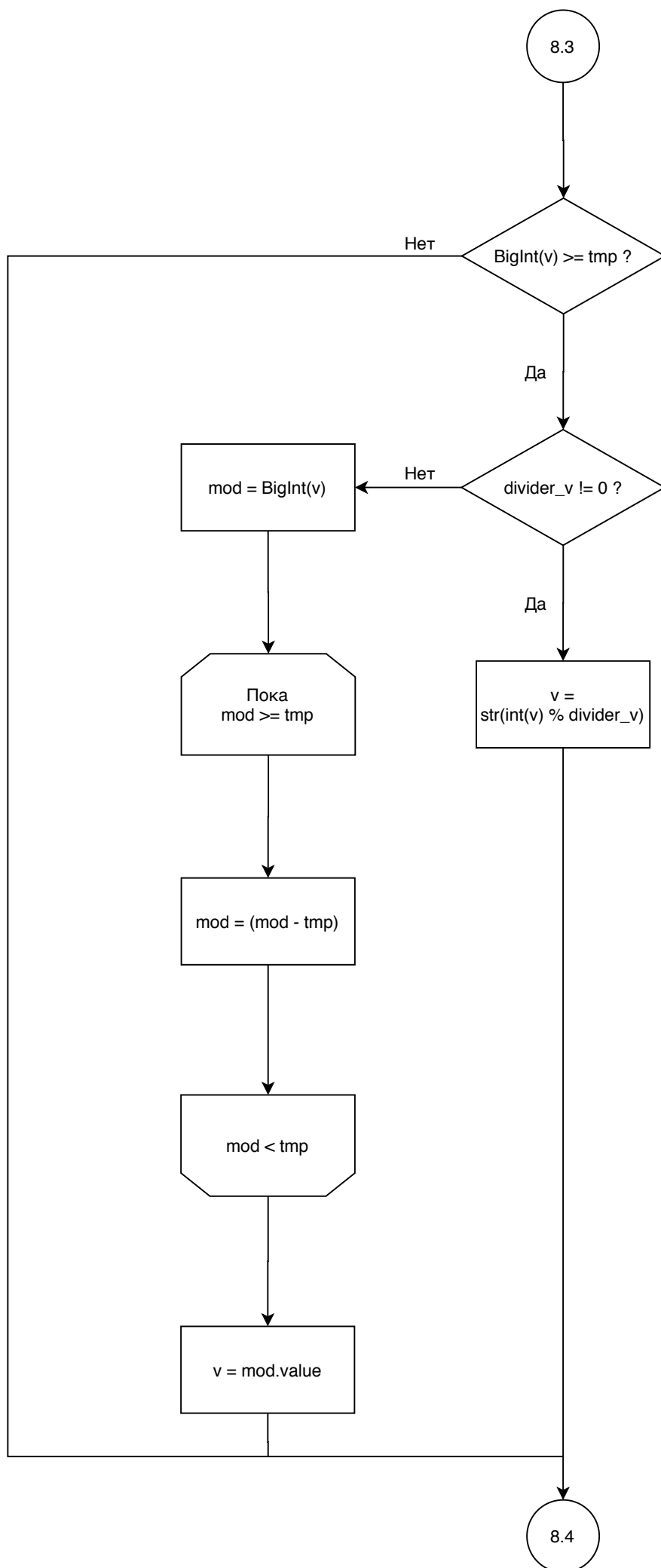
6.3

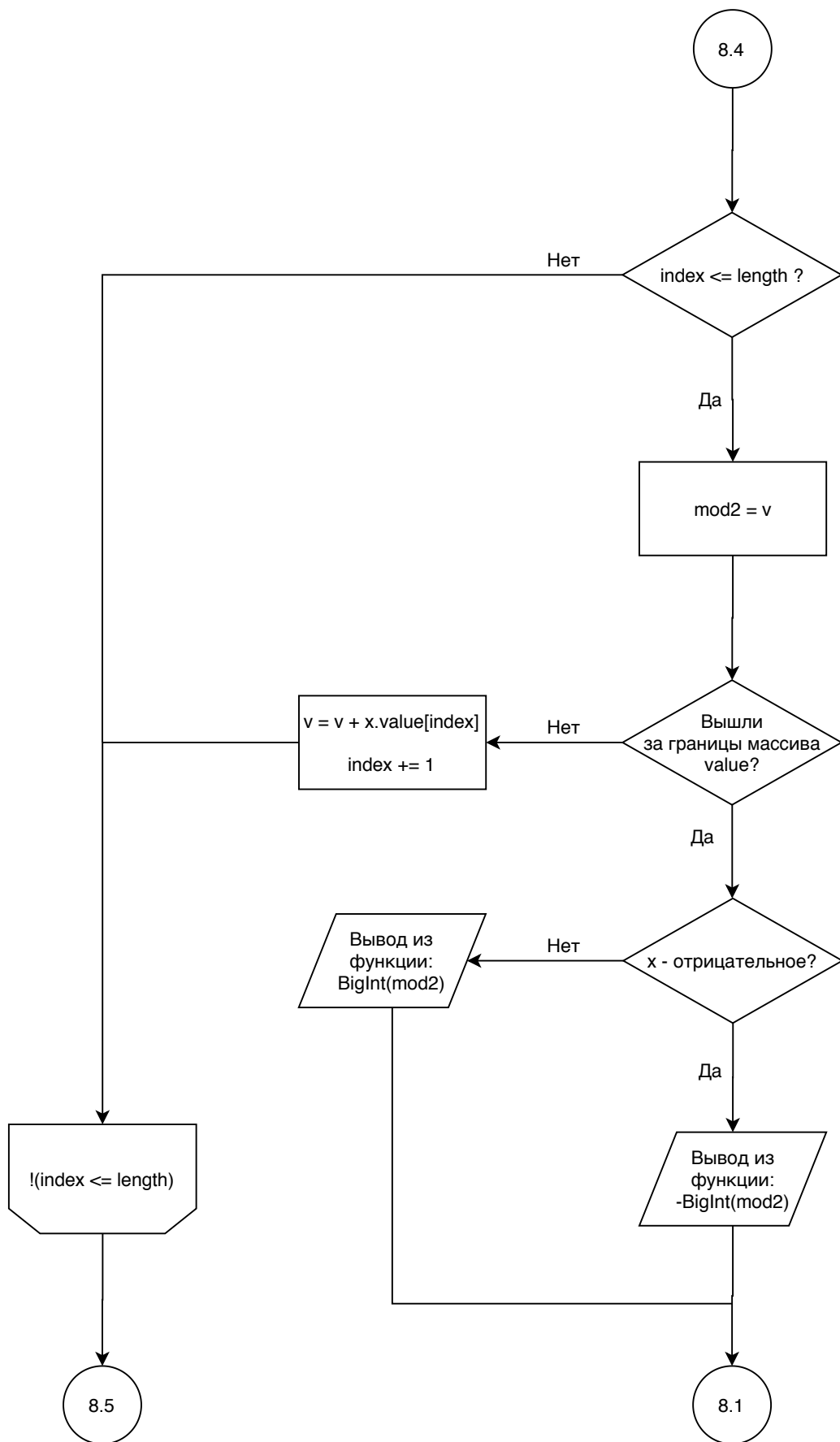


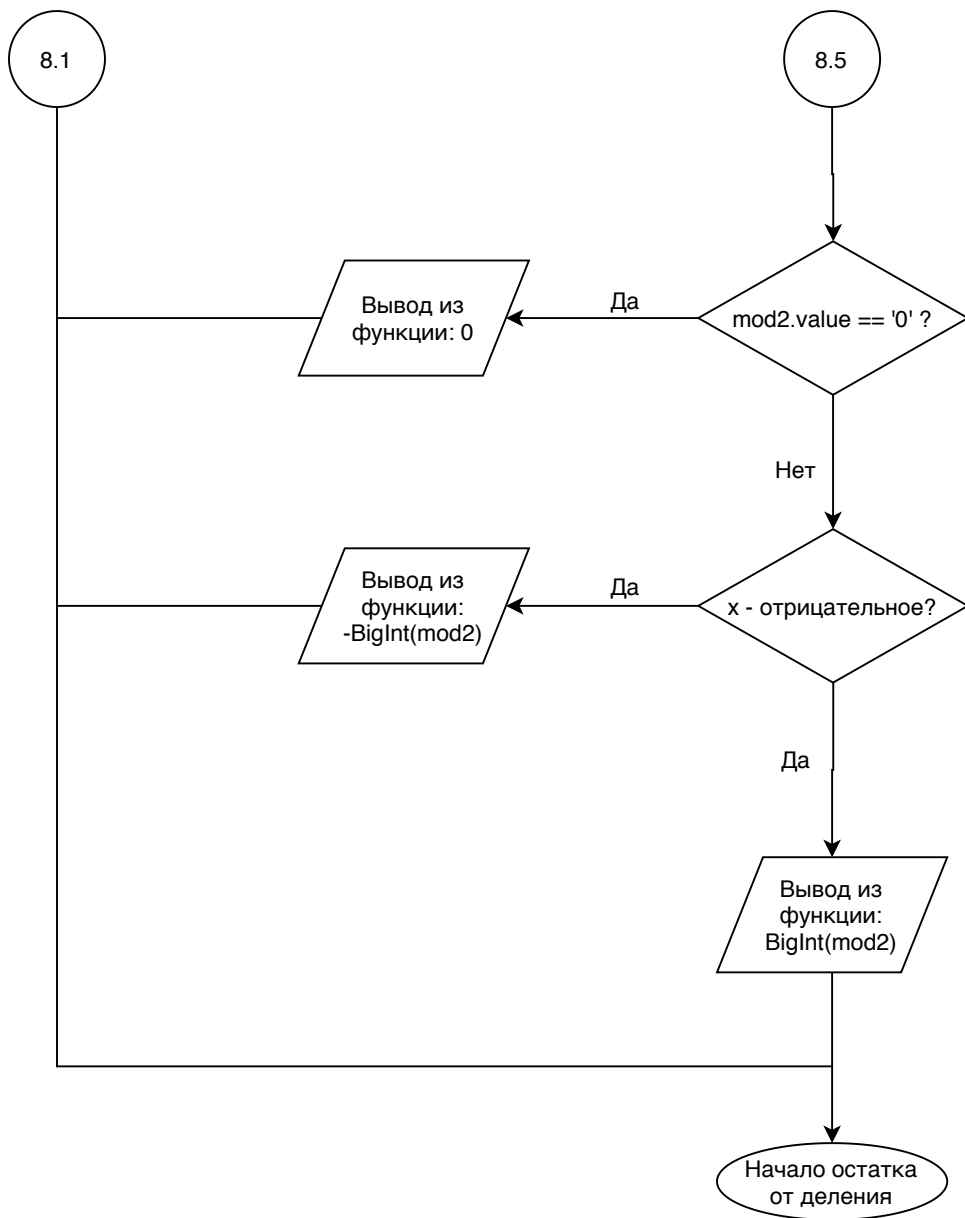












8 Исходный код программы

8.1 Исходный код bigint.py

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  from sys import setrecursionlimit
5
6  setrecursionlimit(1500)  # Максимальный стек рекурсии
7
8
9  class BigInt(object):
10     """Класс работы с большими целыми числами"""
11     is_neg = False  # Флаг отрицательности числа
12     value = ''      # Число в виде строки
13
14     def __init__(self, x=0):
15         """Без аргументов конструктор задает значение равное нулю.
16         Если 'x' - число, конструктор заполняет экземпляр этим числом.
17         Если 'x' - строка, которая могла бы быть числом, конструктор заполня
18         ет экземпляр этим числом.
19         Если 'x' - экземпляр того же класса, конструктор копирует его содерж
20         имое в новый экземпляр."""
21         self.value = '0'
22         # Если в конструктор передано целое число
23         if isinstance(x, int):
24             self.is_neg = x < 0
25             self.value = str(x if x >= 0 else -x)
26         # Если в конструктор передана строка
27         elif isinstance(x, str):
28             # Если вдруг пришла пустая строка, пропускаем, оставляем value =
29             0
30             if len(x):
31                 self.is_neg = x[0] == '-'
32                 # Значением будет все, после минуса, если он был. И убираем
33                 ведущие нули, если они были
34                 self.value = x[self.is_neg:].lstrip('0')
35                 # Проверяем, является ли строка числом, если нет, value = 0
36                 if not self.value.isdigit():
37                     self.value = '0'
38                 if self.value == '0':
```

```

35         self.is_neg = False
36     # Если в конструктор передан экземпляр того же класса, копируем его
    содержимое
37     elif isinstance(x, BigInt):
38         self.value = x.value
39         self.is_neg = x.is_neg
40
41     # Является ли число четным
42     def is_even(self):
43         """Является ли число четным"""
44         return not (int(self.value[-1]) & 1)
45
46     # Перегрузка числа по модулю
47     def __abs__(self):
48         return BigInt(self.value)
49
50     def bipow(self, n):
51         """Возведение числа в степень 'n'"""
52         # Любое число в степени 0 = 1
53         if not n:
54             return BigInt(1)
55         if n & 1:
56             return BigInt(self.bipow(n - 1)) * self
57         tmp = BigInt(self.bipow(n // 2))
58         return BigInt(tmp * tmp)
59
60     def birt(self, n):
61         """Вычисление корня степени 'n' из числа"""
62         # Корень извлекать можем только из положительного числа
63         if (n < 0) or self.is_neg:
64             return None
65         if n == 1:
66             return BigInt(self)
67         length = (len(self.value) + 1) // 2
68         index = 0
69         v = [0] * length
70         while index < length:
71             v[index] = 9
72             while BigInt(''.join(str(x) for x in v)).bipow(n) > self and v[
index]:
73                 v[index] -= 1
74                 index += 1

```

```

75         v = ''.join(str(x) for x in v).lstrip('0')
76         return BigInt('-' + v) if self.is_neg else BigInt(v)
77
78     # Перегрузка перевода в bool
79     def __bool__(self):
80         return self.value != '0'
81
82     # Перегрузка x < y
83     def __lt__(self, other):
84         self_len = len(self.value) # Запоминаем длину первого числа
85         self_other = len(other.value) # Запоминаем длину второго числа
86         # Если знаки одинаковые, то проверяем значения
87         if self.is_neg == other.is_neg:
88             # Если длины не равны
89             if self_len != self_other:
90                 # Меньше число с меньшей длиной для положительных и с
91                 # большей длиной для отрицательных
92                 return (self_len < self_other) ^ self.is_neg
93             i = 0
94             # Ищем разряд, в котором значения отличаются
95             while (i < self_len and self.value[i] == other.value[i]):
96                 i += 1
97             # Если разряд найден, то меньше число с меньшей цифрой для
98             # положительных и с большей цифрой для отрицательных, иначе числа равны
99             return (i < self_len) and ((self.value[i] < other.value[i]) ^
100 self.is_neg)
101         return self.is_neg # Знаки разные, если число отрицательное, то оно
102         # меньше, если положительное, то больше
103
104     # Перегрузка x <= y
105     def __le__(self, other):
106         return self < other or self == other
107
108     # Перегрузка x == y
109     def __eq__(self, other):
110         return (self.value == other.value) and (self.is_neg == other.is_neg)
111
112     # Перегрузка x != y
113     def __ne__(self, other):
114         return not self == other
115
116     # Перегрузка x > y
117     def __gt__(self, other):

```

```

114         return not (self < other or self == other)
115
116     # Перегрузка x >= y
117     def __ge__(self, other):
118         return self > other or self == other
119
120     # Унарный плюс (просто копируем значение числа)
121     def __pos__(self):
122         return BigInt(self)
123
124     # Унарный минус
125     def __neg__(self):
126         return BigInt(self.value if self.is_neg else '-' + self.value)
127
128     # Возврат копии
129     def copy(self):
130         """Возврат копии"""
131         return BigInt(('-' if self.is_neg else '') + self.value)
132
133     # Сложение двух чисел
134     def __add__(self, other):
135         new_int = BigInt()
136         # Если знаки одинаковые, то выполняем сложение
137         if other.is_neg == self.is_neg:
138             num2 = other.value # Запоминаем значение второго операнда
139             self_len = len(self.value) # Длина первого операнда
140             other_len = len(num2) # Длина второго операнда
141             # Длина суммы равна максимуму из двух длин + 1 из-за возможного
переноса разряда
142             length = max(self_len, other_len) + 1
143             res = [0] * length
144             for i in range(length - 1):
145                 j = length - 1 - i
146                 # Выполняем сложение разрядов
147                 res[j] += int((num2[other_len - 1 - i] if i < other_len else
'0')) + int((self.value[self_len - 1 - i] if i < self_len else '0'))
148                 res[j - 1] = res[j] // 10 # Выполняем перенос в следующий
разряд, если он был
149                 res[j] = res[j] % 10 # Оставляем только единицы от
возможного переноса и превращаем символ в цифру
150                 # Возвращаем результат, учитывая его знак
151         return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for
x in res))

```

```

152     # Если одно из чисел отрицательное, а другое положительное,
    отправляем на вычитание, меняя знак
153     return (self - (-BigInt(other))) if self.is_neg else (other - (-
    BigInt(self)))
154
155     # Вычитание одного числа из другого
156     def __sub__(self, other):
157         # Если числа равны, считать не нужно
158         if self == other:
159             return BigInt(0)
160         # Если оба числа положительные, выполняем вычитание
161         if not self.is_neg and not other.is_neg:
162             self_len = len(self.value) # Запоминаем длину первого числа
163             self_other = len(other.value) # Запоминаем длину второго числа
164             length = max(self_len, self_other) # Длина результата не
    превысит максимума длин чисел
165             is_neg_res = other > self # Определяем знак результата
166             # Массивы аргументов
167             a = [0] * length
168             b = [0] * length
169             res = [0] * length
170             sign = 2 * is_neg_res - 1 # Получаем числовое значение знака
    результата
171             for i in range(length - 1):
172                 a[i] += int(self.value[self_len - 1 - i]) if i < self_len
    else 0 # Формируем
    разряды
173                 b[i] += int(other.value[self_other - 1 - i]) if i <
    self_other else 0 # Из строк
    аргументов
174             b[i + 1] = -is_neg_res # В зависимости от знака занимаем
    или не занимаем
175             a[i + 1] = is_neg_res - 1 # 10 у следующего разряда
176             res[length - 1 - i] += 10 + sign * (b[i] - a[i])
177             res[length - 2 - i] = res[length - 1 - i] // 10
178             res[length - 1 - i] = res[length - 1 - i] % 10
179             # Выполняем операцию с последним разрядом
180             a[length - 1] += (length - 1 < self_len) * int(self.value[0])
181             b[length - 1] += (length - 1 < self_other) * int(other.value[0])
182             # Записываем в строку последний разряд
183             res[0] += sign * (b[length - 1] - a[length - 1])
184             # Возвращаем результат, учитывая его знак
185             return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x
    in res))

```

```

186         return -BigInt(other) - (-BigInt(self)) if self.is_neg and other.
is_neg else self + -BigInt(other)
187
188     # Умножение двух чисел
189     def __mul__(self, other):
190         # Если один из множителей равен нулю, то результат равен нулю
191         if self.value == '0' or other.value == '0':
192             return BigInt(0)
193         self_len = len(self.value) # Запоминаем длину первого числа
194         other_len = len(other.value) # Запоминаем длину второго числа
195         length = self_len + other_len + 1 # Результат влезет в сумму длин +
1 из-за возможного переноса
196         # Флаг отрицательности результата - отрицательный, если числа разных
знаков
197         is_neg_res = self.is_neg ^ other.is_neg
198         if length < 10: # Число небольшое, можно по нормальному
199             res = int(self.value) * int(other.value)
200             return BigInt(-res if is_neg_res else res)
201         else: # Умножаем в столбик
202             # Массивы аргументов
203             a = [0] * length
204             b = [0] * length
205             res = [0] * length
206             # Заполняем массивы инверсной записью чисел (с ведущими нулями)
207             for i in range(length):
208                 a[i] = int(self.value[self_len - 1 - i]) if i < self_len
else 0
209                 b[i] = int(other.value[other_len - 1 - i]) if i < other_len
else 0
210             # Выполняем умножение "в столбик"
211             for i in range(self_len):
212                 for j in range(other_len):
213                     res[length - 1 - (i + j)] += a[i] * b[j]
214                     res[length - 1 - (i + j + 1)] += res[length - 1 - (i + j
))] // 10
215                     res[length - 1 - (i + j)] %= 10
216             # Возвращаем результат, учитывая его знак
217             return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for
x in res))
218
219     # Деление одного числа на другое
220     def __truediv__(self, other):
221         value1 = self.value # Запоминаем значение первого числа

```



```

222     value2 = other.value # Запоминаем значение второго числа
223     if value2 == '0':
224         raise ZeroDivisionError # Нельзя делить на ноль
225     if value1 == '0':
226         return BigInt(0) # А вот ноль делить можно на всё, кроме нуля,
но смысл
227     if value2 == '1':
228         return BigInt(-BigInt(self) if other.is_neg else BigInt(self))
# Делить на 1 можно, но
смысл?
229     zeroes = 0
230     while value2[len(value2) - 1 - zeroes] == '0':
231         zeroes += 1
232     if zeroes >= len(value1):
233         return BigInt(0)
234     # Избавляемся от общих нулей в конце чисел
235     if zeroes:
236         value1 = value1[:len(value1) - zeroes]
237         value2 = value2[:len(value2) - zeroes]
238     is_neg_res = self.is_neg ^ other.is_neg # Считаем знак числа
239     tmp = BigInt(value2)
240     divider_length = len(value2) # Запоминаем длину делителя
241     # Если длина больше 8, то обнуляем делитель, иначе переводим строку
в long
242     # Можно не обнулять, но мы думаем, что Python не умеет в большие
числа
243     divider_v = 0 if divider_length > 8 else int(value2)
244     length = len(value1) # Получаем длину делимого
245     index = 0 # Стартуем с нулевого индекса
246     div = '' # Строка результата деления
247     v = '' # Строка подчисла (которое делится на делитель в столбик)
248     # Формируем начальное число для деления
249     while BigInt(v) < tmp and index < length:
250         v = v + value1[index]
251         index += 1
252     mod = None
253     while True:
254         count = 0 # Результат деления подчисла на делитель
255         # Если можем разделить, то делим
256         if BigInt(v) >= tmp:
257             # Если не входит в long, то делим с помощью вычитания
258             if divider_length > 8:
259                 mod = BigInt(v)
260                 while mod >= tmp:

```

```

261         mod = (mod - tmp).copy()
262         count += 1
263         v = mod.value
264     else:
265         mod = int(v)
266         count = mod // divider_v
267         v = str(mod % divider_v)
268     # Если не делили, то добавили ноль к результату, иначе добавили
результат деления
269     div = div + (str(count) if count else '0')
270     if index <= length:
271         try: # Тот самый ноль, лучше не спрашивать
272             v = v + value1[index]
273         except IndexError:
274             v = v + '0'
275         index += 1 # Формируем новое значение для подчисла
276     if not (index <= length):
277         break
278     # Возвращаем результат учитывая знак и возможное равенство нулю
279     return BigInt('-' + div if is_neg_res and div != '0' else div)
280
281     # Обработка для выходных данных
282     def __str__(self):
283         return str('-' if self.is_neg else '') + self.value
284
285     # Остаток от деления
286     def __mod__(self, other):
287         if other.value == '0':
288             return None
289         if self.value == '0' or other.value == "1":
290             return BigInt(0)
291         # Если числа меньше 9, можно посчитать по нормальному
292         if len(self.value) < 9 and len(other.value) < 9:
293             res = int(self.value) % int(other.value)
294             return BigInt(-res if self.is_neg else res)
295         tmp = BigInt(other.value)
296         divider_length = len(other.value) # запоминаем длину делителя
297         # Если длина больше 8, то обнуляем long'овый делитель, иначе
переводим строку в long
298         divider_v = 0 if divider_length > 8 else int(other.value)
299         length = len(self.value)
300         index = 0

```


8.2 Исходный код main.py

```
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  from sys import argv
5
6  from PyQt5 import QtCore, QtGui, QtWidgets
7
8  import about
9  import helpme
10 import ui
11 from bigint import GCD, BigInt
12
13
14 # Класс главной формы
15 class LongArithmeticCalc(QtWidgets.QMainWindow, ui.Ui_MainWindow):
16     def __init__(self):
17         super().__init__()
18         self.setupUi(self)
19
20         self.error_message = 'Ошибка ввода :('
21
22         # Events кнопок меню описания и выбора файла
23         self.addition.clicked.connect(self.addition_clicked)
24
25         #
26         +
27         self.subtraction.clicked.connect(self.subtraction_clicked)
28
29         #
30         -
31         self.multiplication.clicked.connect(self.multiplication_clicked)
32
33         #
34         *
35         self.division.clicked.connect(self.division_clicked)
36
37         #
38         /
39         self.exponentiation.clicked.connect(self.exponentiation_clicked)
40
41         #
42         степень
43         self.root.clicked.connect(self.root_clicked)
44
45         #
46         корень
47         self.gdc.clicked.connect(self.gdc_clicked)
48
49         #
50         НОД
```

```

30         self.remainder.clicked.connect(self.remainder_clicked)
#
OCT
31         self.about.clicked.connect(self.about_clicked)
32         self.helpme.clicked.connect(self.helpme_clicked)
33
34     def get_nums(self):
35         return self.first_num_edit.toPlainText(), self.second_num_edit.
toPlainText()
36
37     def data_validation(self, a):
38         try:
39             int(a)
40             return True
41         except ValueError:
42             return False
43
44     def addition_clicked(self): # +
45         first_num, second_num = self.get_nums()
46         if self.data_validation(first_num) and self.data_validation(
second_num):
47             res = str(BigInt(first_num) + BigInt(second_num))
48             self.result.setText(res)
49         else:
50             self.result.setText(self.error_message)
51
52     def subtraction_clicked(self): # -
53         first_num, second_num = self.get_nums()
54         if self.data_validation(first_num) and self.data_validation(
second_num):
55             res = str(BigInt(first_num) - BigInt(second_num))
56             self.result.setText(res)
57         else:
58             self.result.setText(self.error_message)
59
60     def multiplication_clicked(self): # *
61         first_num, second_num = self.get_nums()
62         if self.data_validation(first_num) and self.data_validation(
second_num):
63             res = str(BigInt(first_num) * BigInt(second_num))
64             self.result.setText(res)
65         else:

```

```

66         self.result.setText(self.error_message)
67
68     def division_clicked(self): # /
69         first_num, second_num = self.get_nums()
70         if self.data_validation(first_num) and self.data_validation(
second_num):
71             res = str(BigInt(first_num) / BigInt(second_num))
72             self.result.setText(res)
73         else:
74             self.result.setText(self.error_message)
75
76     def exponentiation_clicked(self): # степень
77         first_num, second_num = self.get_nums()
78         if self.data_validation(first_num) and self.data_validation(
second_num):
79             res = str(BigInt(first_num).bipow(int(second_num)))
80             self.result.setText(res)
81         else:
82             self.result.setText(self.error_message)
83
84     def root_clicked(self): # корень
85         first_num, second_num = self.get_nums()
86         if self.data_validation(first_num) and self.data_validation(
second_num):
87             res = str(BigInt(first_num).birt(int(second_num)))
88             self.result.setText(res)
89         else:
90             self.result.setText(self.error_message)
91
92     def gcd_clicked(self): # НОД
93         first_num, second_num = self.get_nums()
94         if self.data_validation(first_num) and self.data_validation(
second_num):
95             res = str(GCD(BigInt(first_num), BigInt(second_num)))
96             self.result.setText(res)
97         else:
98             self.result.setText(self.error_message)
99
100     def remainder_clicked(self): # Ост
101         first_num, second_num = self.get_nums()

```

```

102         if self.data_validation(first_num) and self.data_validation(
second_num):
103             res = str(BigInt(first_num) % BigInt(second_num))
104             self.result.setText(res)
105         else:
106             self.result.setText(self.error_message)
107
108     def about_clicked(self): # about
109         about_dialog = about()
110         about_dialog.exec_()
111
112     def helpme_clicked(self): # Помощь
113         helpme_dialog = helpme()
114         helpme_dialog.exec_()
115
116
117 class about(QtWidgets.QDialog, about.Ui_Dialog):
118     def __init__(self):
119         super().__init__()
120         self.setupUi(self)
121         pixmap = QtGui.QPixmap('./img/math_logo.png')
122         self.ejik.setPixmap(pixmap)
123         self.resize(pixmap.width(), pixmap.height())
124
125
126 class helpme(QtWidgets.QDialog, helpme.Ui_helpme):
127     def __init__(self):
128         super().__init__()
129         self.setupUi(self)
130
131
132 def main():
133     app = QtWidgets.QApplication(argv)
134     main_window = LongArithmeticCalc()
135     main_window.show()
136     app.exec_()
137
138
139 if __name__ == "__main__":
140     main()

```
