

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 2021

Содержание

1	Постановка задачи	3
2	Используемые инструменты	4
3	Общая структура библиотеки целых длинных чисел	5
4	Примеры работы библиотеки	7
5	Вывод	9
6	Блок-схема методов библиотеки	10
7	Исходный код программы	29
	Список литературы	42

1 Постановка задачи

Составьте алгоритм (в виде блок-схемы) и напишите (на любом языке программирования) соответствующую ему программу, позволяющую выполнять арифметические операции (сложение, вычитание, умножение и деление) над длинными целыми числами.

2 Используемые инструменты

Для решения вышеуказанной задачи были использованы следующие инструменты:

- Основным ЯП был выбран Python версии 3.9.2;
- Для компиляции программы в бинарный файл .exe использован конвертер файлов Auto PY to EXE, который использует для своей работы PyInstaller.

3 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.
- Вычитание целых длинных чисел.
Программная реализация представляет собой вычитание чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания.
- Умножение целых длинных чисел.
Программная реализация представляет собой умножение чисел в «столбик». Данный способ реализации был выбран по причине простоты его работы и написания.
- Целочисленное деление целых длинных чисел.
Программная реализация представляет собой деление чисел с помощью метода вычитания. Данный способ реализации был выбран по причине простоты его работы и написания.

Класс «BigInt» содержит в себе два основных поля:

1. Поле хранения числа «value».
Представляет собой переменную типа строка, в котором 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 следующие способами:

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

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

- 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
```

- 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
```

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

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

4 Примеры работы библиотеки

В качестве примера работы будут использоваться прямые вызовы методов класса «BigInt».

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()
```

- Выполним сложение:

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

Вывод: 8641975319135802468

- Выполним вычитание:

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

Вывод: -111111110111111110

- Выполним умножение:

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

Вывод: -12193263121170553265523548251112635269

- Выполним целочисленное деление:

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

Вывод: -8

- Выполним деление на ноль:

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

Вывод: *ZeroDivisionError*

- Выполним деление нуля:

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

Вывод: 0

- Выполним умножение на ноль:

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

Вывод: 0

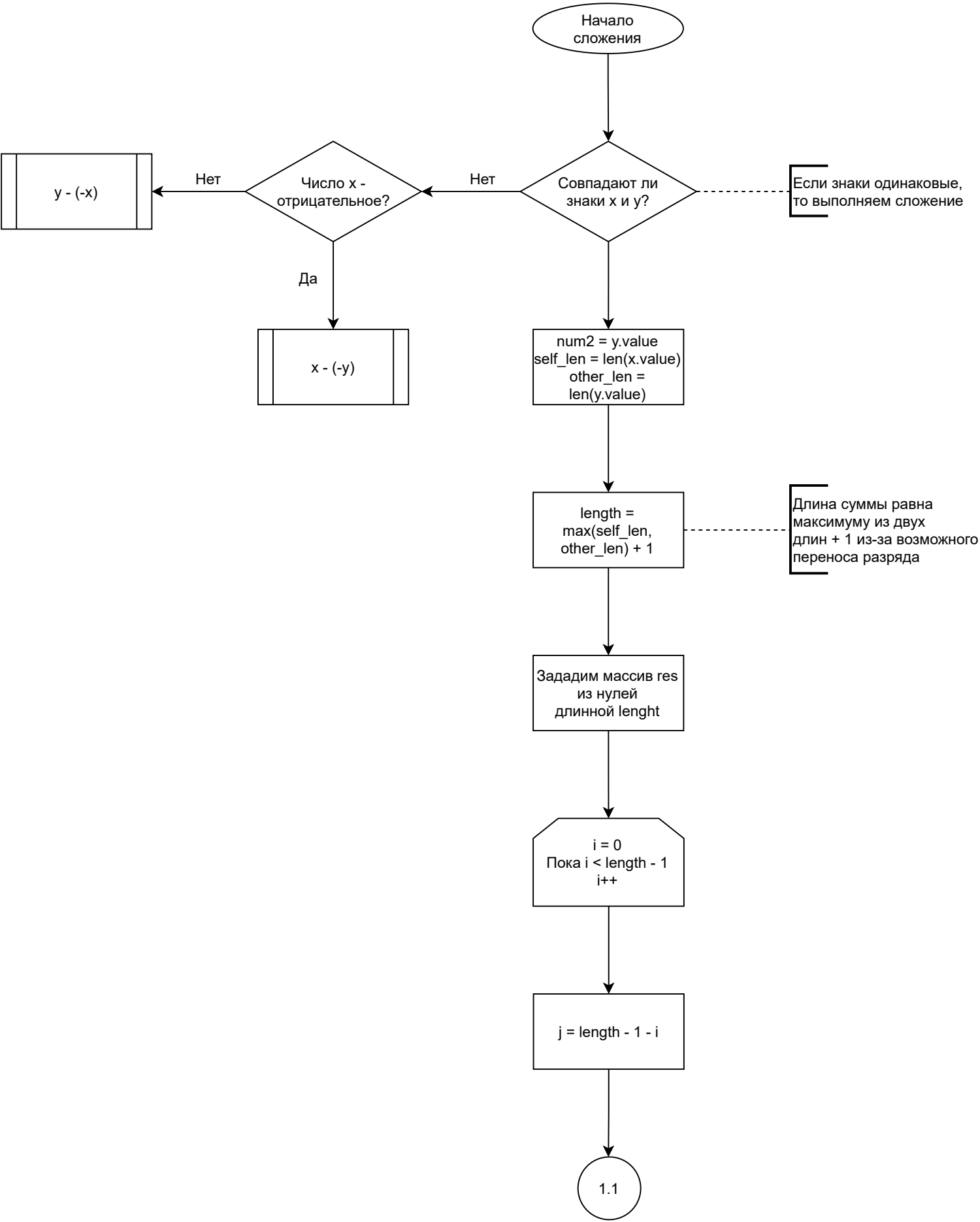
5 Вывод

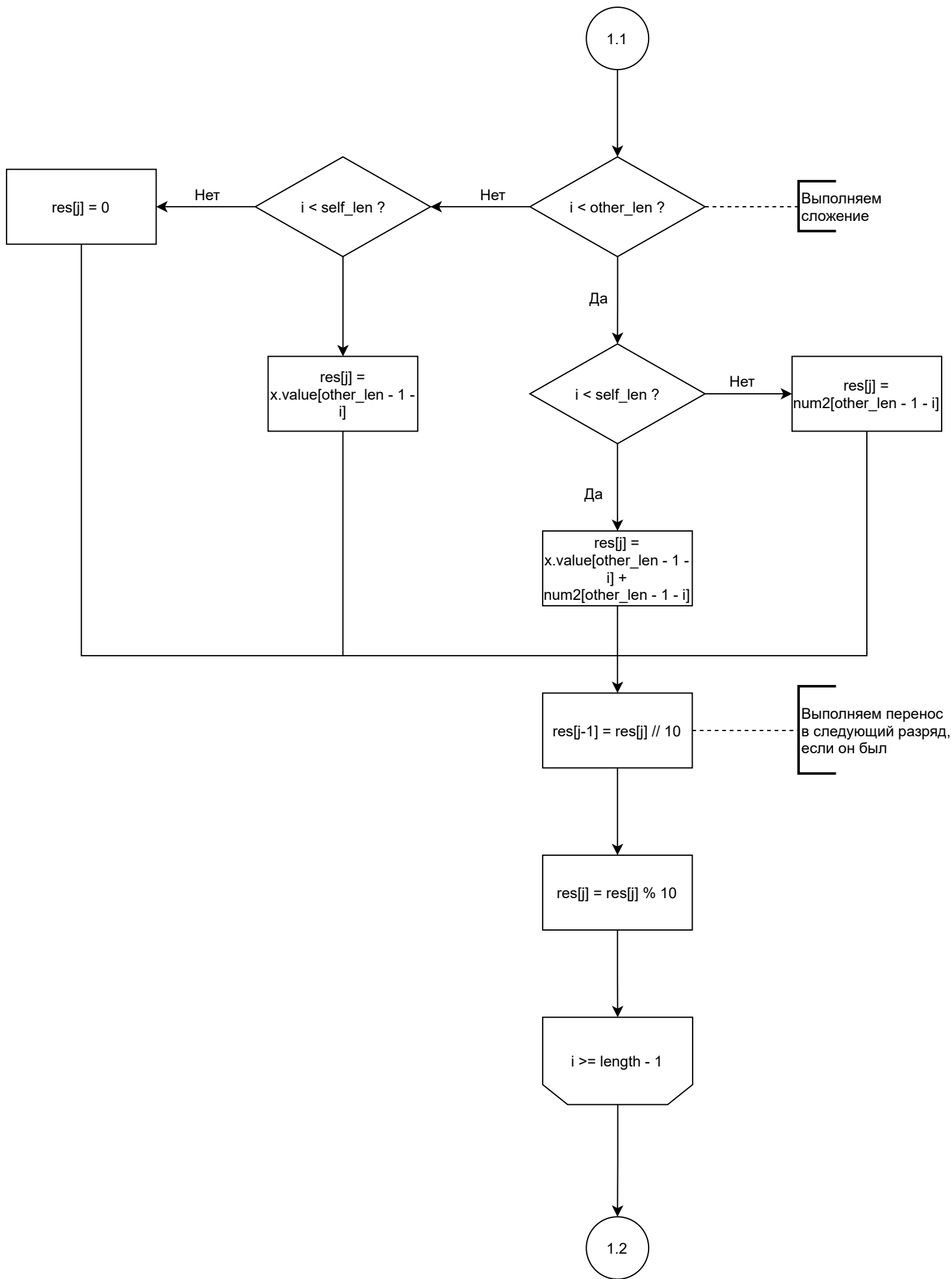
Мною был составлен алгоритм (в виде блок-схемы) и написана на языке Python программа, позволяющая выполнять арифметические операции (сложение, вычитание, умножение и деление) над длинными целыми числами.

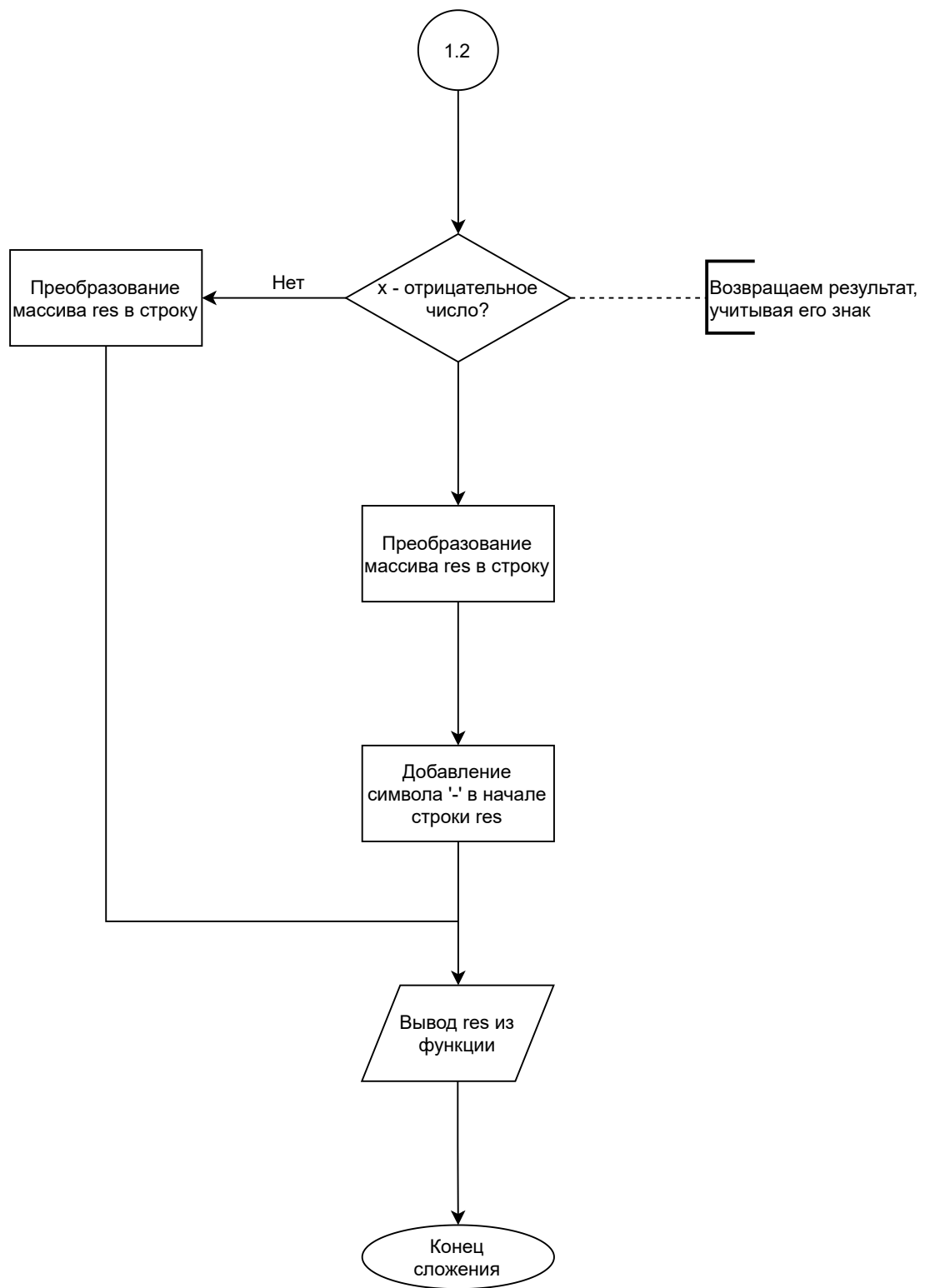
6 Блок-схема методов библиотеки

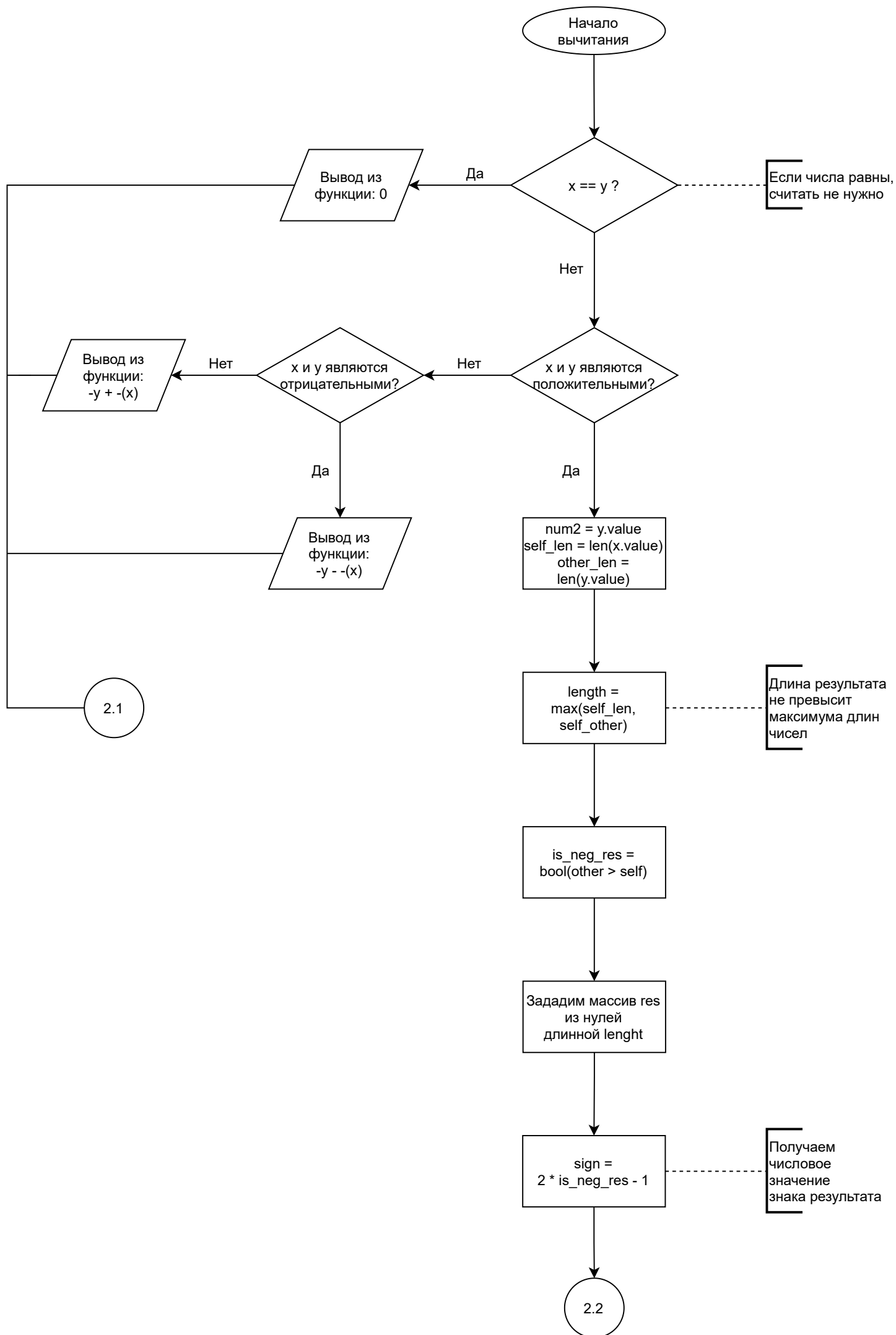
Ниже представлены блок-схемы методов в следующем порядке:

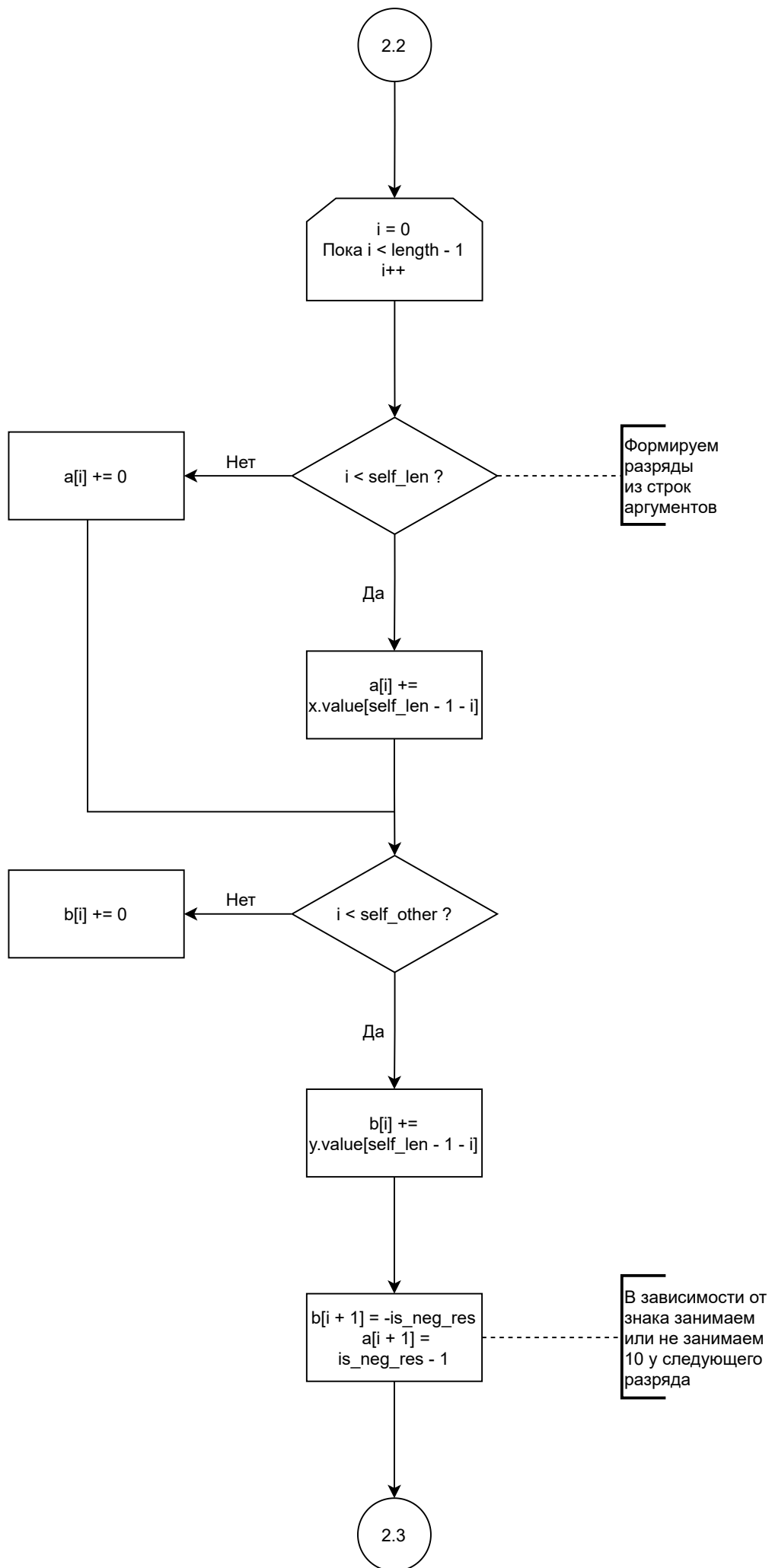
1. Сложение;
2. Вычитание;
3. Умножение;
4. Деление.

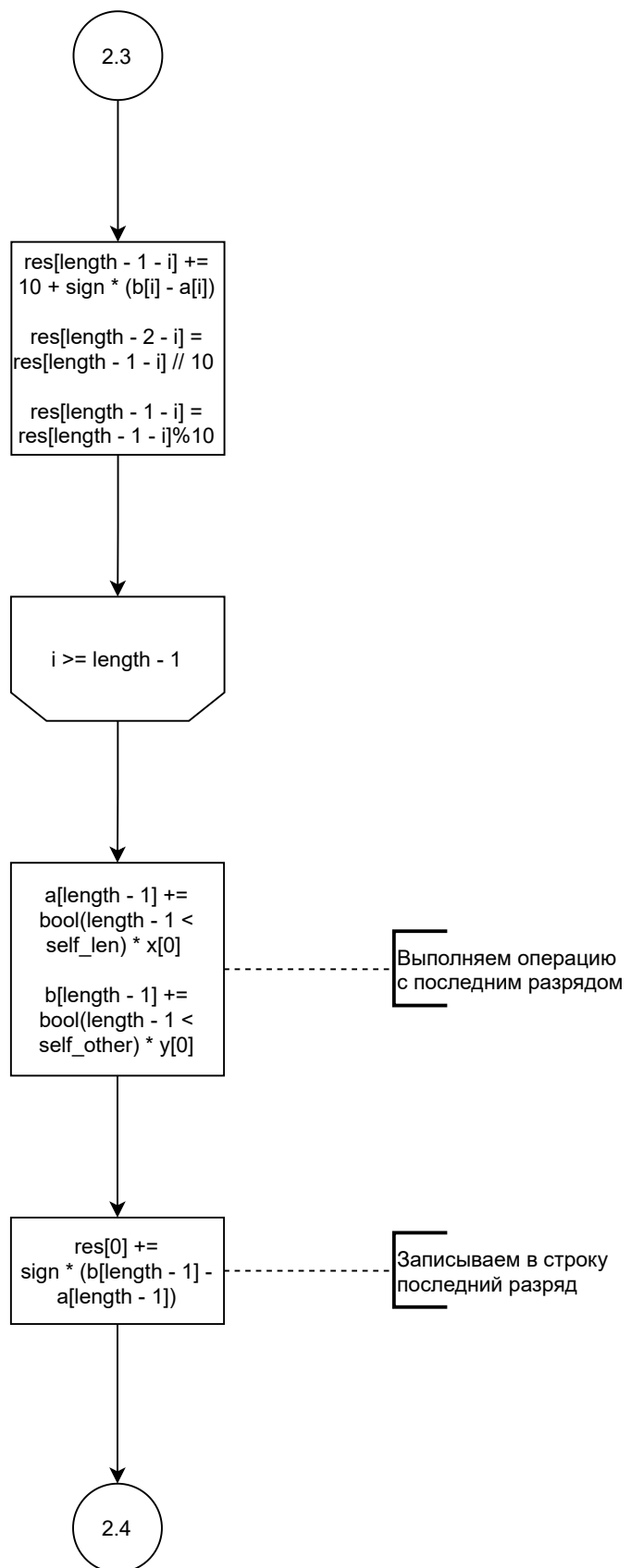


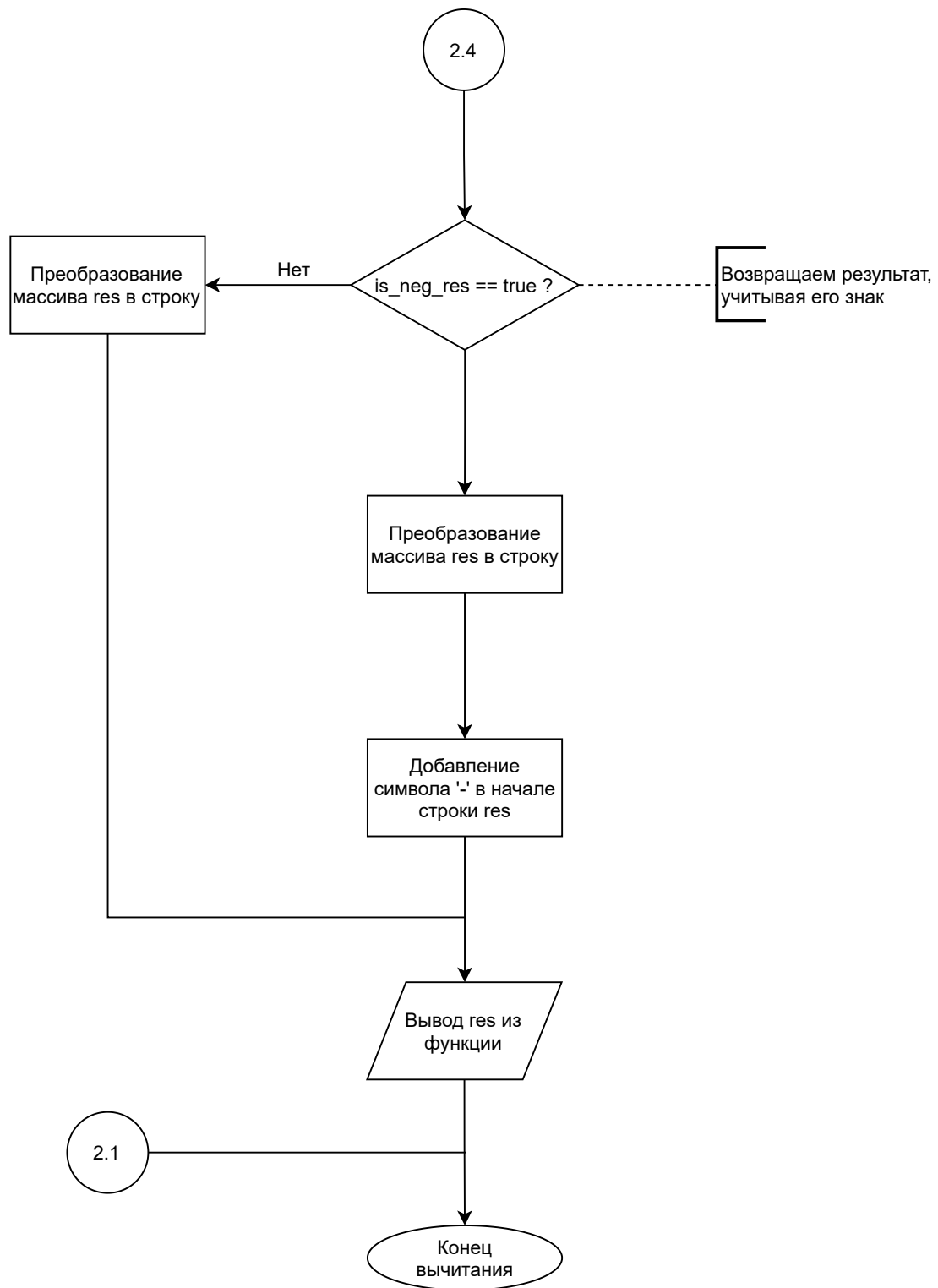


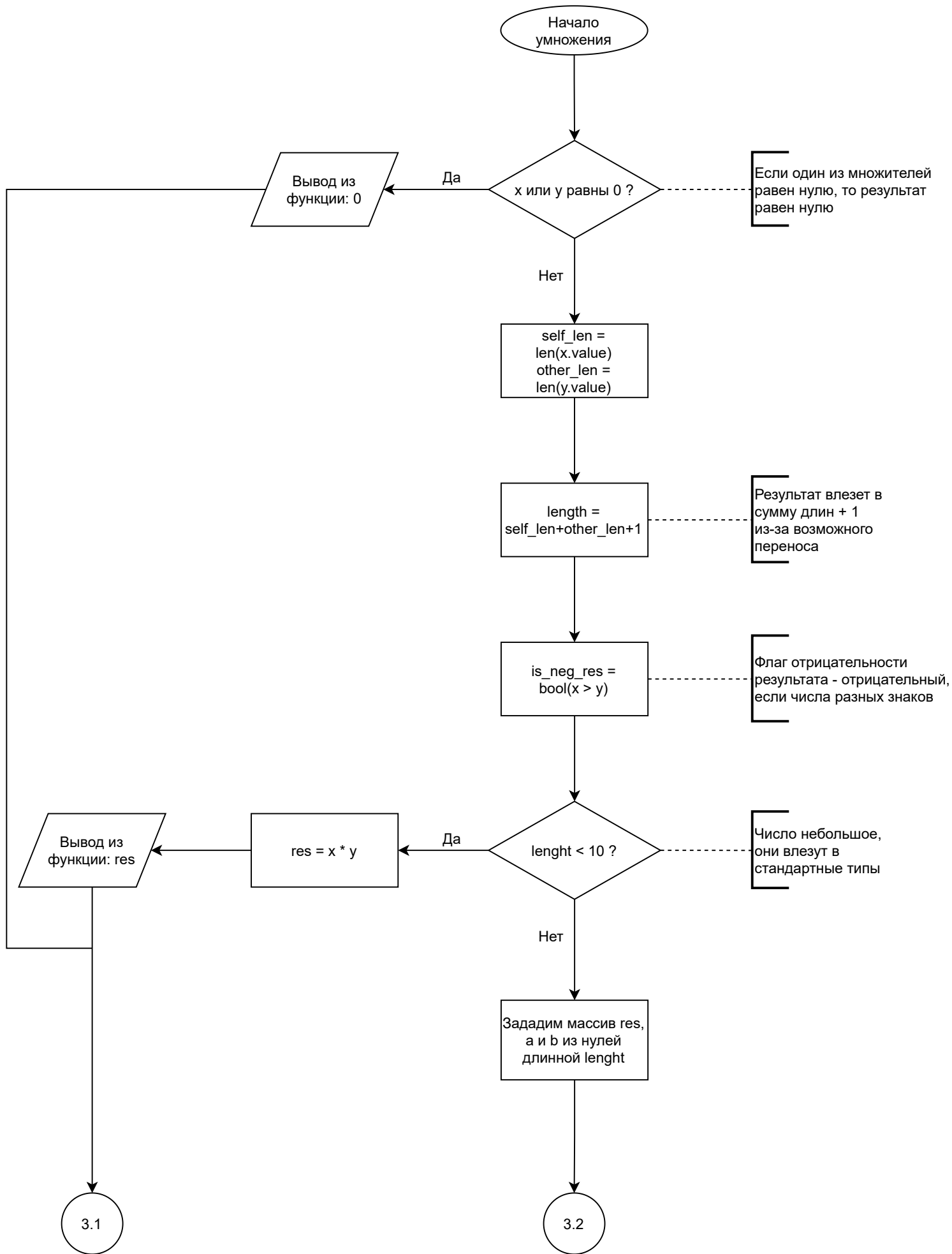


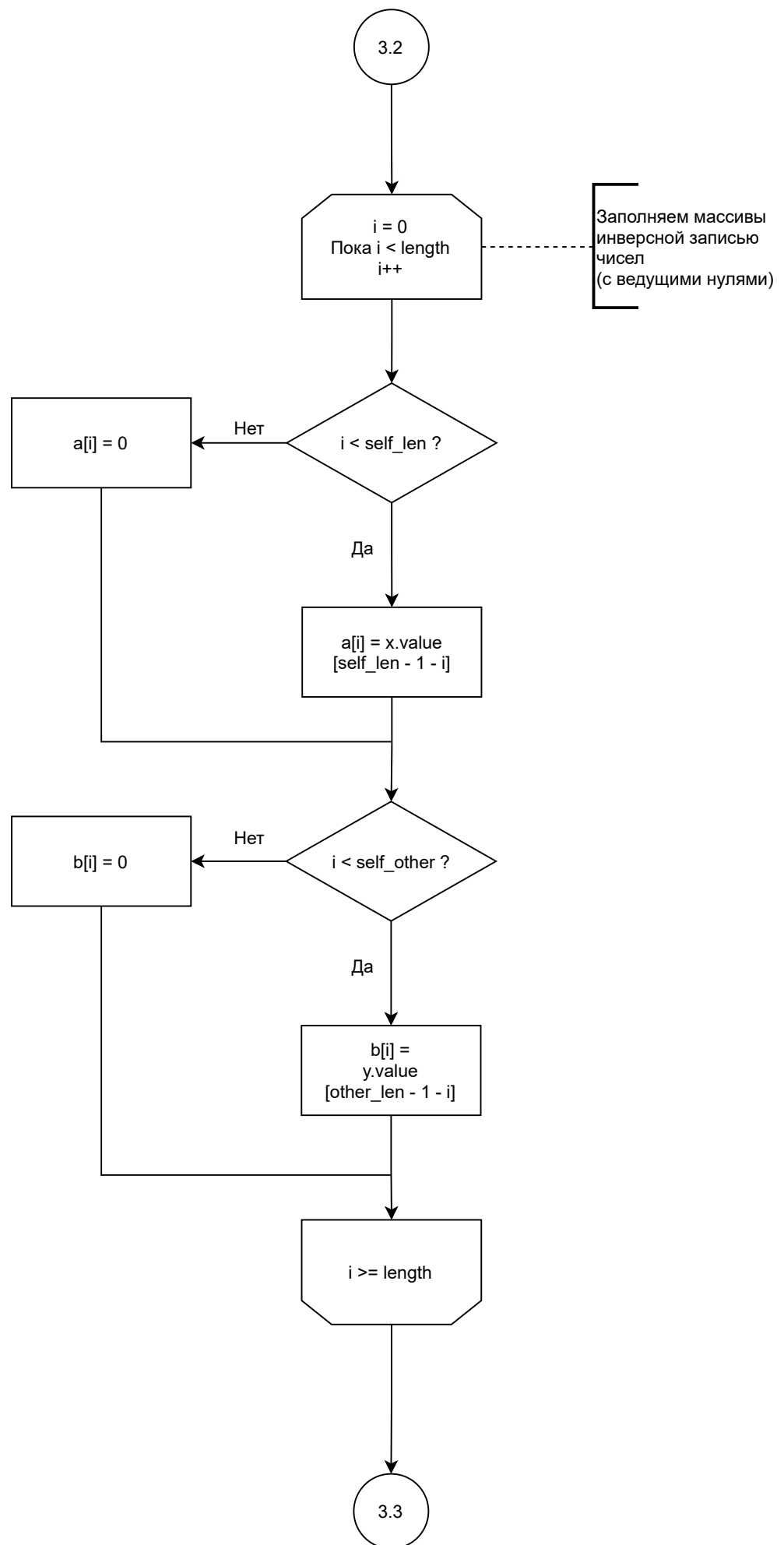


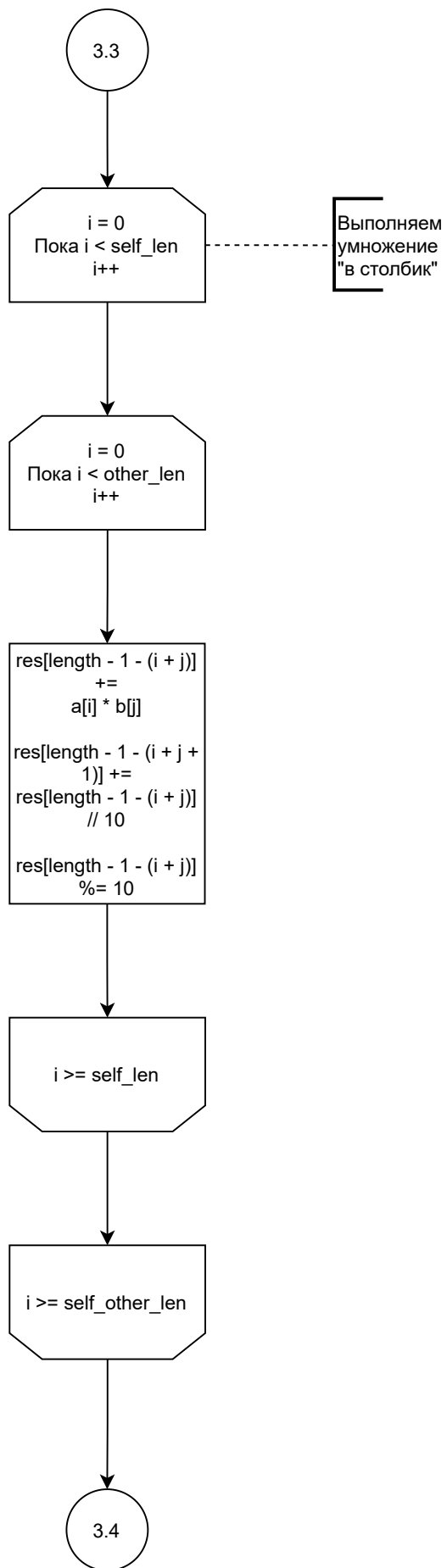


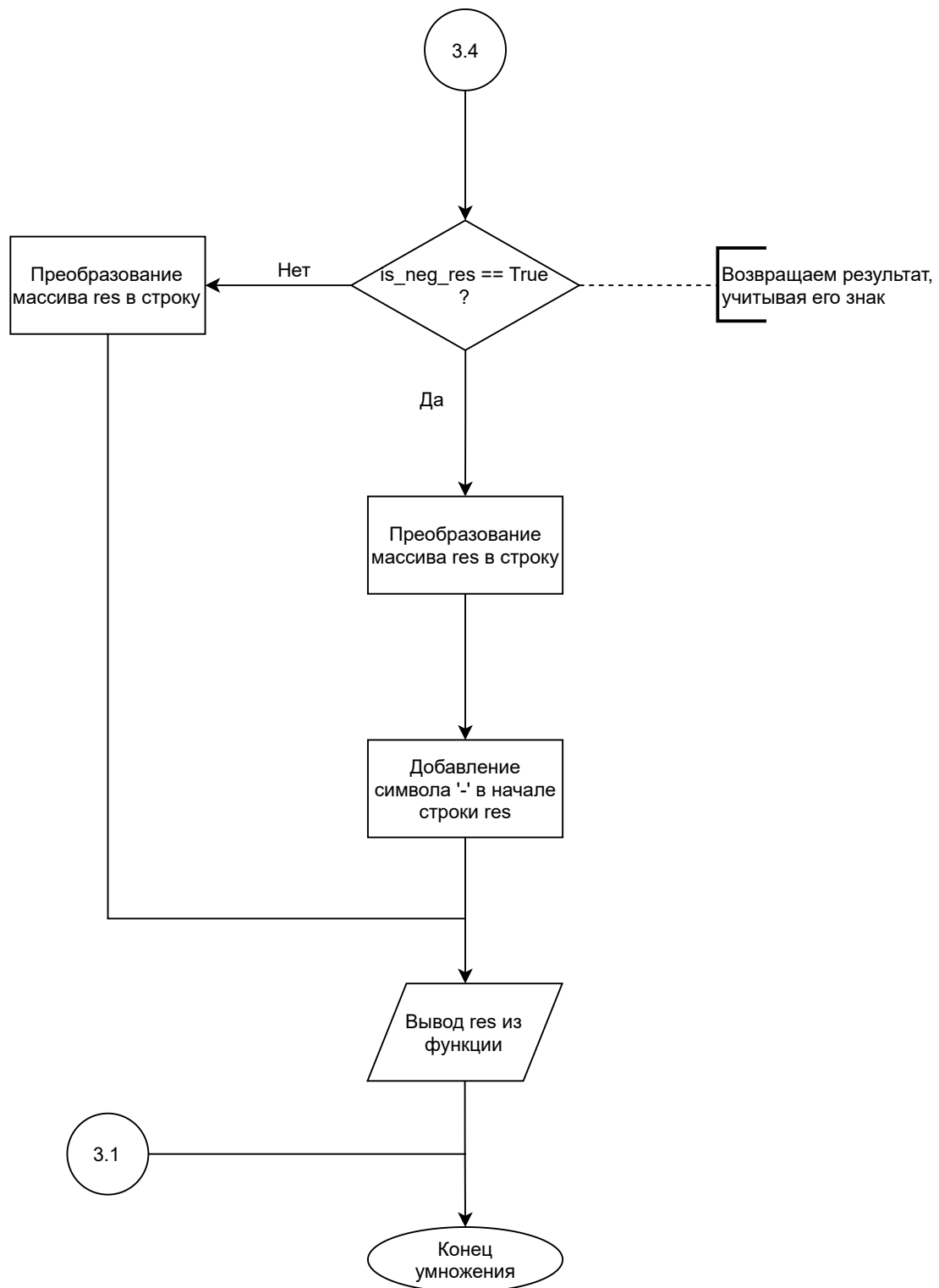


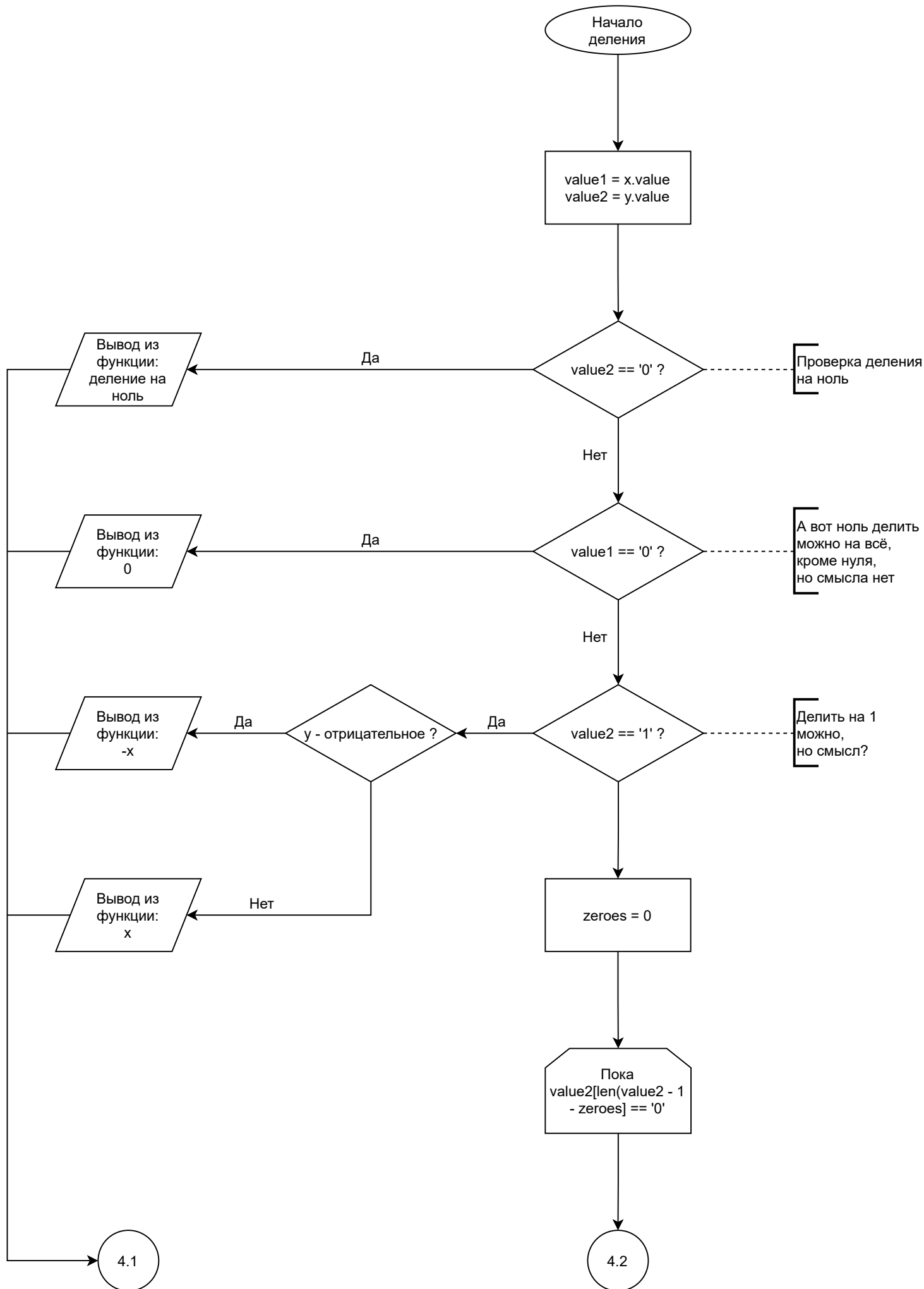


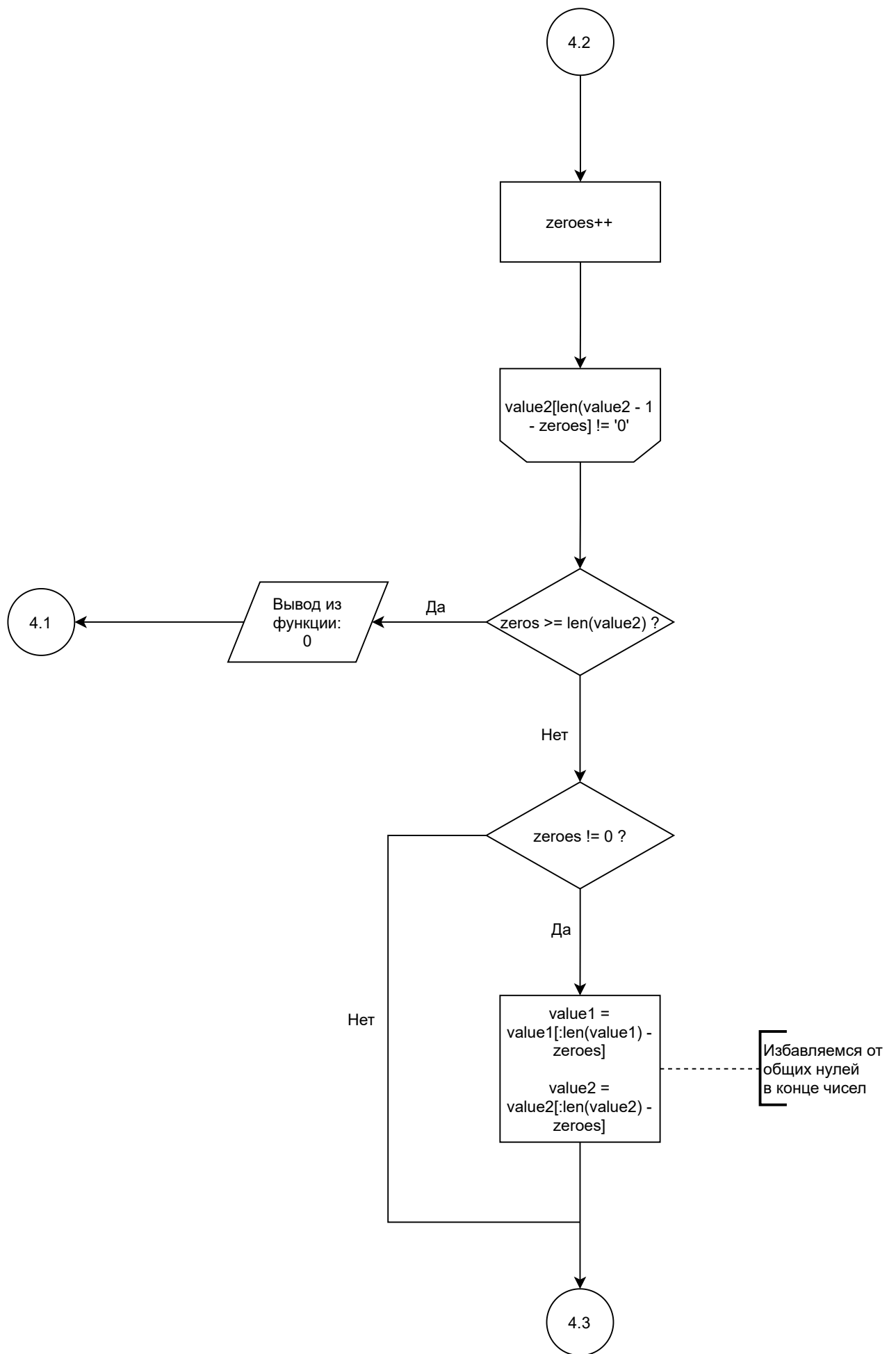


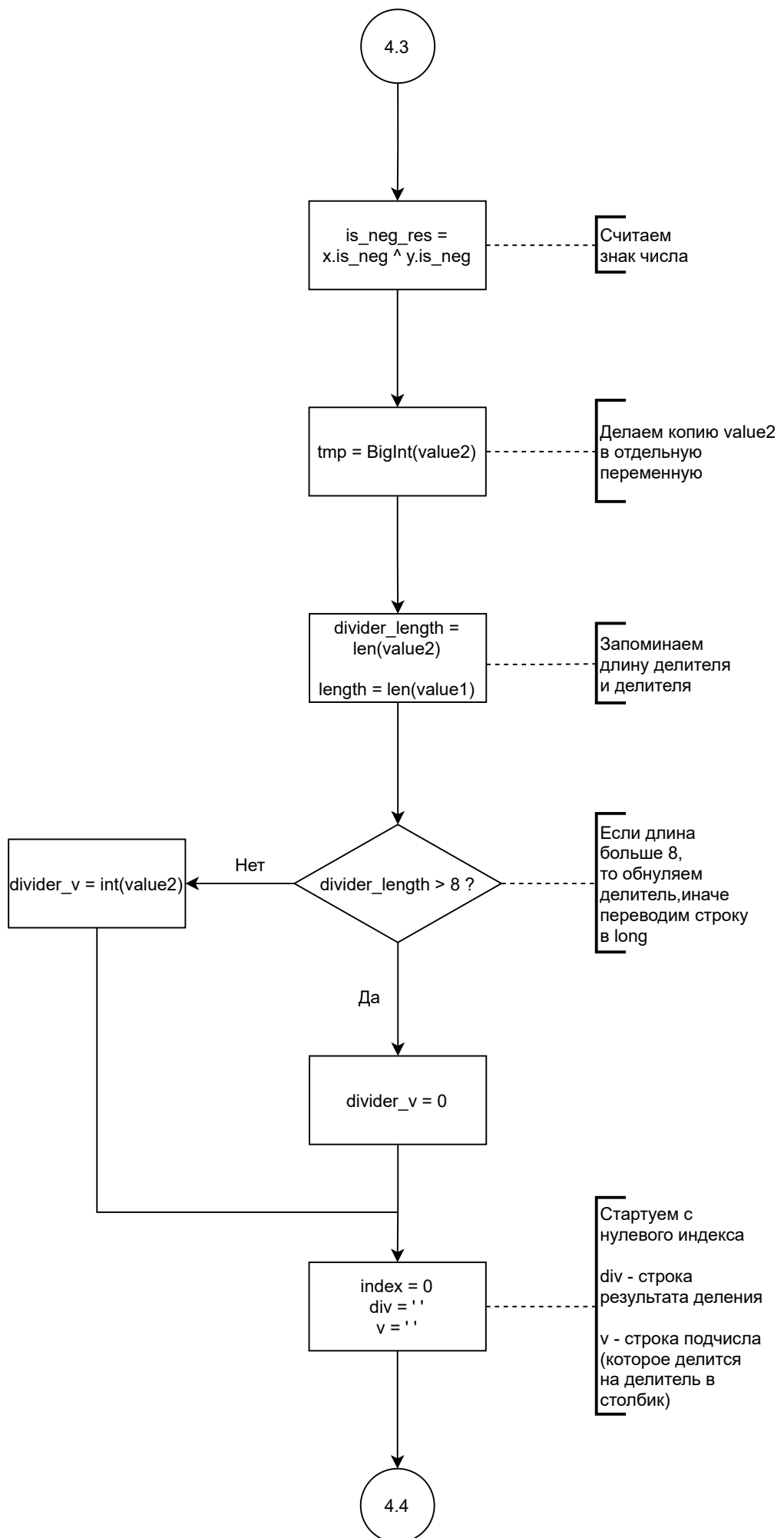


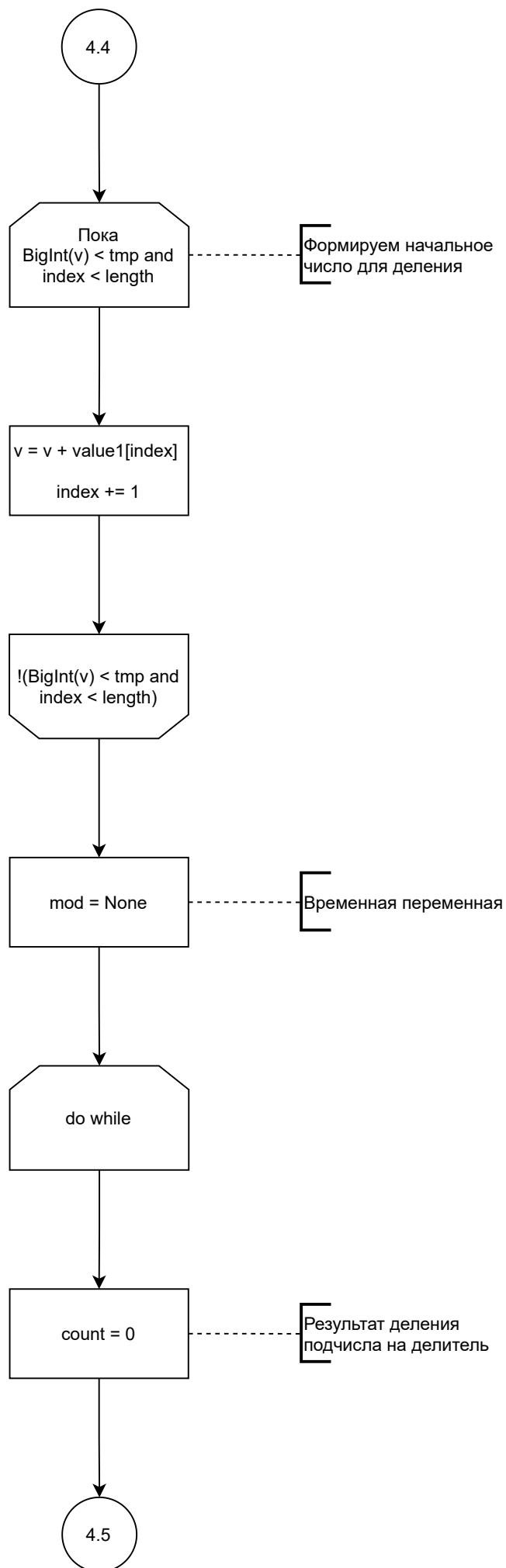


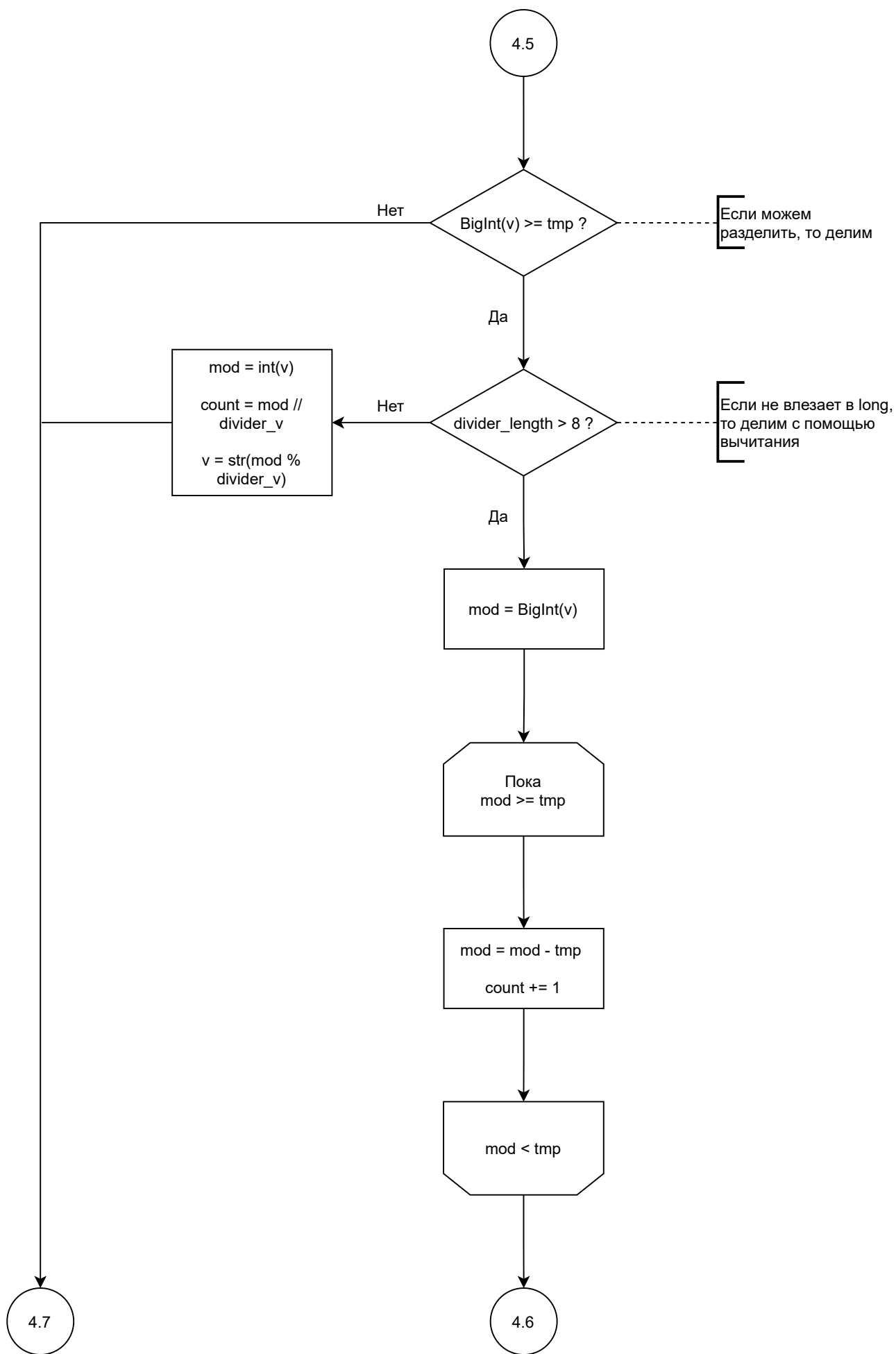


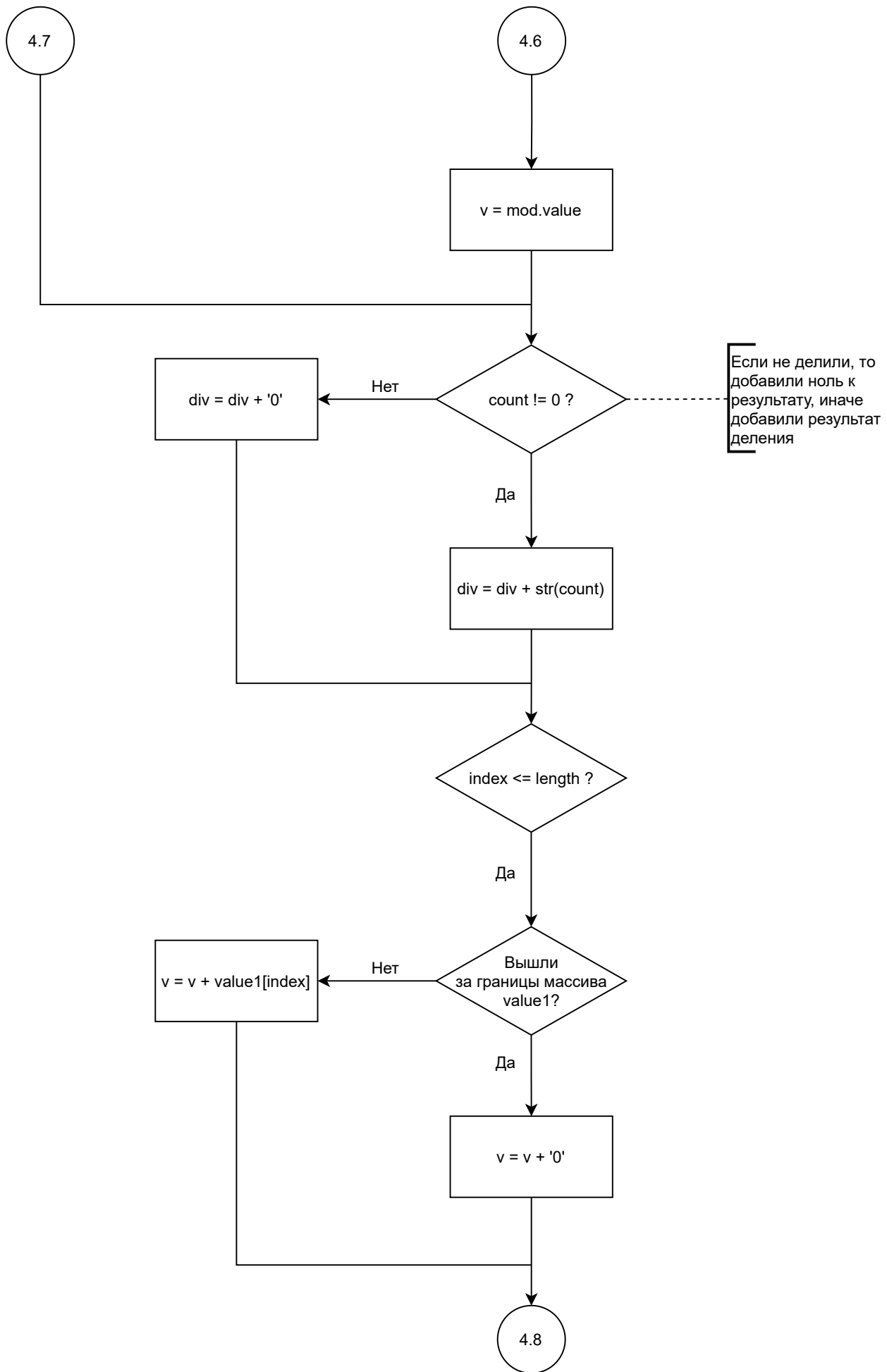


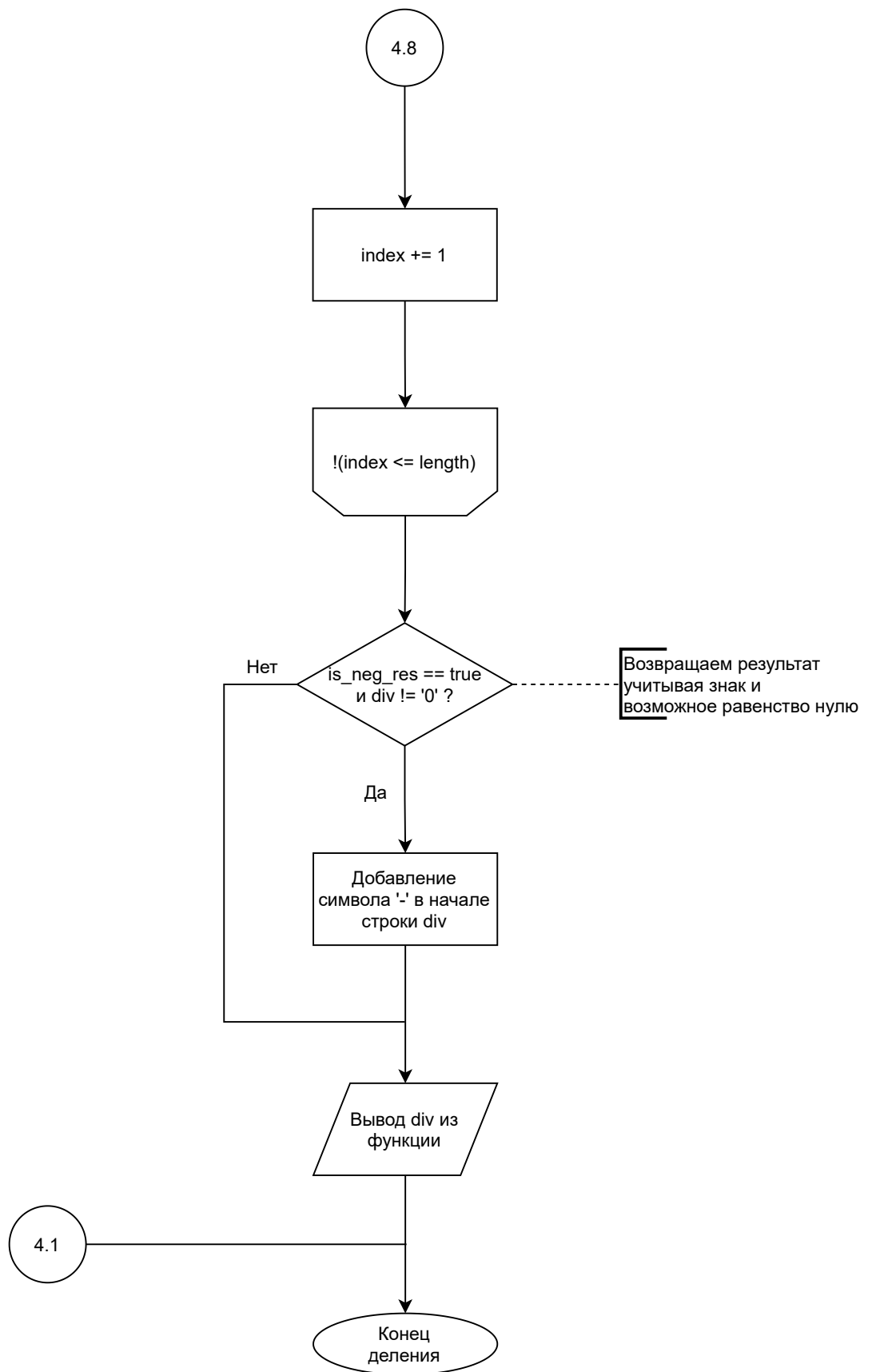












7 Исходный код программы

```
1 # -*- coding: utf-8 -*-
2
3 import time
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         """Безаргументов конструктор задает значение равно нулю.
16         Если 'x' - число, конструктор заполняет экземпляр этим числом.
17         Если 'x' - строка, которая могла бы быть числом, конструктор заполняет экзе
18             мпляр этим числом.
19         Если 'x' - экземпляр того же класса, конструктор копирует его содержимое в но
20             вый 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':
39                     self.is_neg = False
40             # Если в конструктор передан экземпляр того же класса, копируем его
41             # 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 n < 0:
54             return None
55         if not n:
56             return BigInt(1)
57         # if n & 1:
58         # return BigInt(self.bipow(n - 1)) * self
59         # tmp = BigInt(self.bipow(n // 2))
60         # return BigInt(tmp * tmp)
61         res = BigInt(1)
62         for _ in range(n + 1):
63             res *= self
64         return res
65
66     def birt(self, n):
67         """Вычисление корня степени 'n' из числа"""
68         # Корень извлекать можем только из положительного числа
69         if (n < 0) or self.is_neg:
70             return None
71         if n == 1:
72             return self
73         length = (len(self.value) + 1) // 2
74         index = 0
75         v = [0] * length
76         while index < length:
77             v[index] = 9

```

```

78         while BigInt('').join(str(x) for x in v)).bipow(n) > self and v[
            index]:
79             v[index] -= 1
80             index += 1
81     v = ''.join(str(x) for x in v).lstrip('0')
82     return BigInt('-' + v) if self.is_neg else BigInt(v)
83
84     # Перегрузка перевода в bool
85     def __bool__(self):
86         return self.value != '0'
87
88     # Перегрузка x < y
89     def __lt__(self, other):
90         if isinstance(other, int):
91             other = BigInt(other)
92         self_len = len(self.value) # Запоминаем длину первого числа
93         self_other = len(other.value) # Запоминаем длину второго числа
94         # Если знаки одинаковые, то проверяем значения
95         if self.is_neg == other.is_neg:
96             # Если длины не равны
97             if self_len != self_other:
98                 # Меньше число с меньшей длиной для положительных и с
99                 # большей длиной для отрицательных
100                 return (self_len < self_other) ^ self.is_neg
101             i = 0
102             # Ищем разряд, в котором значения отличаются
103             while (i < self_len and self.value[i] == other.value[i]):
104                 i += 1
105             # Если разряд найден, то меньше число с меньшей цифрой для
106             # положительных и с большей цифрой для отрицательных, иначе
107             # числа равны
108             return (i < self_len) and ((self.value[i] < other.value[i]) ^
109                                     self.is_neg)
110         return self.is_neg # Знаки разные, если число отрицательное, то оно
111         # меньше, если положительное, то больше
112
113     # Перегрузка x <= y
114     def __le__(self, other):
115         return self < other or self == other
116
117     # Перегрузка x == y
118     def __eq__(self, other):
119         if isinstance(other, int):

```

```

115         other = BigInt(other)
116         return (self.value == other.value) and (self.is_neg == other.is_neg)
117
118     # Перегрузка x != y
119     def __ne__(self, other):
120         return not self == other
121
122     # Перегрузка x > y
123     def __gt__(self, other):
124         return not (self < other or self == other)
125
126     # Перегрузка x >= y
127     def __ge__(self, other):
128         return self > other or self == other
129
130     # Унарный плюс (просто копируем значение числа)
131     def __pos__(self):
132         return self
133
134     # Унарный минус
135     def __neg__(self):
136         return BigInt(self.value if self.is_neg else '-' + self.value)
137
138     # Число в бинарный вид (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
139     def to_bin(self):
140         return bin(int(self.value))
141
142     # Битовый сдвиг вправо (x » y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
143     def __rshift__(self, n):
144         if n < 0:
145             raise ValueError
146         self_bin = self.to_bin()
147         if n >= len(self_bin) - 2 or (self == 0):
148             return BigInt(0)
149         return BigInt(int(self_bin[:len(self_bin) - n], 2))
150
151     # Битовый сдвиг влево (x « y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
152     def __lshift__(self, n):
153         if n < 0:
154             raise ValueError
155         if self == 0:

```



```

156         return BigInt(0)
157     self_bin = self.to_bin()
158     return BigInt(int(self_bin + ('0' * n), 2))
159
160     # Побитовое И (x & y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
161     def __and__(self, other):
162         if isinstance(other, int):
163             other = BigInt(other)
164             self_bin = self.to_bin()[2:]
165             other_bin = other.to_bin()[2:]
166             self_len = len(self_bin)
167             other_len = len(other_bin)
168             if self_len > other_len:
169                 other_bin = other_bin.zfill(self_len)
170             elif self_len < other_len:
171                 self_bin = self_bin.zfill(other_len)
172             res = int('0b' + ''.join(['1' if (x, y) == ('1', '1') else '0' for x,
173                                     y in zip(self_bin, other_bin)]), 2)
174             return BigInt(res)
175
176     # Побитовое ИЛИ (x | y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
177     def __or__(self, other):
178         if isinstance(other, int):
179             other = BigInt(other)
180             self_bin = self.to_bin()[2:]
181             other_bin = other.to_bin()[2:]
182             self_len = len(self_bin)
183             other_len = len(other_bin)
184             if self_len > other_len:
185                 other_bin = other_bin.zfill(self_len)
186             elif self_len < other_len:
187                 self_bin = self_bin.zfill(other_len)
188             res = int('0b' + ''.join(['0' if (x, y) == ('0', '0') else '1' for x,
189                                     y in zip(self_bin, other_bin)]), 2)
190             return BigInt(res)
191
192     # Возврат копии
193     def copy(self):
194         """Возвраткопии"""
195         return BigInt(('-' if self.is_neg else '') + self.value)

```

```

195     # Сложение двух чисел
196     def __add__(self, other):
197         if isinstance(other, int):
198             other = BigInt(other)
199         # Если знаки одинаковые, то выполняем сложение
200         if other.is_neg == self.is_neg:
201             num2 = other.value # Запоминаем значение второго операнда
202             self_len = len(self.value) # Длина первого операнда
203             other_len = len(num2) # Длина второго операнда
204             # Длина суммы равна максимуму из двух длин + 1 из-за возможного
                переноса разряда
205             length = max(self_len, other_len)
206             res = [0] * (length + 1)
207             for i in range(length):
208                 j = length - i
209                 # Выполняем сложение разрядов
210                 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'))
211                 res[j - 1] = res[j] // 10 # Выполняем перенос в следующий
                    разряд, если он был
212                 res[j] = res[j] % 10 # Оставляем только единицы от возможного
                    переноса и превращаем символ в цифру
213                 # Возвращаем результат, учитывая его знак
214                 return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for x
                    in res))
215             # Если одно из чисел отрицательное, а другое положительное,
                отправляем на вычитание, меняя знак
216             return (self - (-BigInt(other))) if self.is_neg else (other - (-
                BigInt(self)))
217
218     # Вычитание одного числа из другого
219     def __sub__(self, other):
220         if isinstance(other, int):
221             other = BigInt(other)
222         # Если числа равны, считать не нужно
223         if self == other:
224             return BigInt(0)
225         # Если оба числа положительные, выполняем вычитание
226         if not self.is_neg and not other.is_neg:
227             self_len = len(self.value) # Запоминаем длину первого числа
228             self_other = len(other.value) # Запоминаем длину второго числа
229             length = max(self_len, self_other) - 1 # Длина результата не
                превысит максимума длин чисел

```

```

230         is_neg_res = other > self # Определяем знак результата
231         # Массивы аргументов
232         new_length = length + 1
233         a = [0] * new_length
234         b = [0] * new_length
235         res = [0] * new_length
236         sign = 2 * is_neg_res - 1 # Получаем числовое значение знака
                                     результата
237         for i in range(length):
238             a[i] += int(self.value[self_len - 1 - i]) if i < self_len
                                     else 0 # Формируем
                                     разряды
239             b[i] += int(other.value[self_other - 1 - i]) if i <
                                     self_other else 0 # Из строк
                                     аргументов
240             b[i + 1] = -is_neg_res # В зависимости от знака занимаем или
                                     не занимаем
241             a[i + 1] = is_neg_res - 1 # 10 у следующего разряда
242             res[length - i] += 10 + sign * (b[i] - a[i])
243             res[length - 1 - i] = res[length - i] // 10
244             res[length - i] = res[length - i] % 10
245         # Выполняем операцию с последним разрядом
246         a[length] += (length < self_len) * int(self.value[0])
247         b[length] += (length < self_other) * int(other.value[0])
248         # Записываем в строку последний разряд
249         res[0] += sign * (b[length] - a[length])
250         # Возвращаем результат, учитывая его знак
251         return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x
                                     in res))
252     return -BigInt(other) - (-BigInt(self)) if self.is_neg and other.
        is_neg else self + -BigInt(other)
253
254     # Умножение двух чисел
255     def __mul__(self, other):
256         if isinstance(other, int):
257             other = BigInt(other)
258         # Если один из множителей равен нулю, то результат равен нулю
259         if self.value == '0' or other.value == '0':
260             return BigInt(0)
261         self_len = len(self.value) # Запоминаем длину первого числа
262         other_len = len(other.value) # Запоминаем длину второго числа
263         length = self_len + other_len # Результат влезет в сумму длин + 1
                                     из-за возможного переноса
264         # Флаг отрицательности результата - отрицательный, если числа разных
                                     знаков

```

```

265         is_neg_res = self.is_neg ^ other.is_neg
266     if length < 10: # Число небольшое, можно по нормальному
267         res = int(self.value) * int(other.value)
268         return BigInt(-res if is_neg_res else res)
269     else: # Умножаем в столбик
270         # Массивы аргументов
271         new_length = length + 1
272         a = [0] * new_length
273         b = [0] * new_length
274         res = [0] * new_length
275         # Заполняем массивы инверсной записью чисел (с ведущими нулями)
276         for i in range(new_length):
277             a[i] = int(self.value[self_len - 1 - i]) if i < self_len else
                0
278             b[i] = int(other.value[other_len - 1 - i]) if i < other_len
                else 0
279         # Выполняем умножение "в столбик"
280         for i in range(self_len):
281             for j in range(other_len):
282                 res[length - (i + j)] += a[i] * b[j]
283                 res[length - (i + j + 1)] += res[length - (i + j)] // 10
284                 res[length - (i + j)] %= 10
285         # Возвращаем результат, учитывая его знак
286         return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x
                in res))
287
288     # Деление одного числа на другое
289     def __truediv__(self, other):
290         if isinstance(other, int):
291             other = BigInt(other)
292         value1 = self.value # Запоминаем значение первого числа
293         value2 = other.value # Запоминаем значение второго числа
294         if value2 == '0':
295             raise ZeroDivisionError # Нельзя делить на ноль
296         if value1 == '0':
297             return BigInt(0) # А вот ноль делить можно на всё, кроме нуля, но
                СМЫСЛ
298         if value2 == '1':
299             return -BigInt(self) if other.is_neg else BigInt(self) # Делить
                на 1 можно, но смысл?
300         zeroes = 0
301         while value2[len(value2) - 1 - zeroes] == '0':
302             zeroes += 1

```

```

303     if zeroes >= len(value1):
304         return BigInt(0)
305     # если у нас 13698 / 1000, то мы можем делить 13 / 1
306     if zeroes:
307         value1 = value1[:len(value1) - zeroes]
308         value2 = value2[:len(value2) - zeroes]
309     is_neg_res = self.is_neg ^ other.is_neg # Считаем знак числа
310     tmp = BigInt(value2)
311     divider_length = len(value2) # Запоминаем длину делителя
312     # Если длина больше 8, то обнуляем делитель, иначе переводим строку
        в long
313     # Можно не обнулять, но мы думаем, что Python не умеет в большие
        числа
314     divider_v = 0 if divider_length > 8 else int(value2)
315     length = len(value1) # Получаем длину делимого
316     index = 0 # Стартуем с нулевого индекса
317     div = ''# Строка результата деления
318     v = ''# Строка подчисла (которое делится на делитель в столбик)
319     index = len(value2)
320     v = value1[:index]
321     mod = None
322     while True:
323         count = 0 # Результат деления подчисла на делитель
324         # Если можем разделить, то делим
325         if BigInt(v) >= tmp:
326             # Если не входит в long, то делим с помощью вычитания
327             if divider_length > 8:
328                 mod = BigInt(v)
329                 while mod >= tmp:
330                     mod = (mod - tmp).copy()
331                     count += 1
332                 v = mod.value
333             else:
334                 mod = int(v)
335                 count = mod // divider_v
336                 v = str(mod % divider_v)
337         # Если не делили, то добавили ноль к результату, иначе добавили
            результат деления
338         div = div + (str(count) if count else '0')
339         if index <= length:
340             try: # Тот самый ноль, лучше не спрашивать
341                 v = v + value1[index]
342             except IndexError:

```

```

343         v = v + '0'
344         index += 1 # Формируем новое значение для подчисла
345         if not (index <= length):
346             break
347     # Возвращаем результат учитывая знак и возможное равенство нулю
348     return BigInt('-' + div if is_neg_res and div != '0' else div)
349
350 # Обработка для выходных данных
351 def __str__(self):
352     return str('-' if self.is_neg else '') + self.value
353
354 # Остаток от деления
355 def __mod__(self, other):
356     if isinstance(other, int):
357         other = BigInt(other)
358     if other.value == '0':
359         return None
360     if self.value == '0' or other.value == "1":
361         return BigInt(0)
362     # Если числа меньше 9, можно посчитать по нормальному
363     if len(self.value) < 9 and len(other.value) < 9:
364         res = int(self.value) % int(other.value)
365         return BigInt(-res if self.is_neg else res)
366     tmp = BigInt(other.value)
367     divider_length = len(other.value) # запоминаем длину делителя
368     # Если длина больше 8, то обнуляем long'овый делитель, иначе
        переводим строку в long
369     divider_v = 0 if divider_length > 8 else int(other.value)
370     length = len(self.value)
371     index = 0
372     mod2 = self.copy()
373     v = ''
374     mod = None
375     while BigInt(v) < tmp and index < length:
376         v = v + self.value[index]
377         index += 1
378     while True:
379         if BigInt(v) >= tmp:
380             if divider_v:
381                 v = str(int(v) % divider_v)
382             else:
383                 mod = BigInt(v)

```

```

384         while mod >= tmp:
385             mod = (mod - tmp).copy()
386             v = mod.value
387         if index <= length:
388             mod2 = v
389             try:
390                 v = v + self.value[index]
391             except IndexError:
392                 break
393             index += 1
394         if not (index <= length):
395             break
396         if isinstance(mod2, BigInt):
397             if mod2.value == '0':
398                 return BigInt(0)
399         res = -BigInt(mod2) if self.is_neg else BigInt(mod2)
400         if self.is_neg ^ other.is_neg:
401             return other + res
402         return res
403
404
405 def GCD(a, b):
406     """Нахождениенаибольшегообщегоделителяучисел'a' и'b'"""
407     if a < 0:
408         a = -a
409     if b < 0:
410         b = -b
411     while b:
412         a, b = b, a % b
413     return a
414
415
416 def binary_GCD(num1, num2):
417     """Нахождениенаибольшегообщегоделителяучисел'a' и'b' (Бинарный алгоритм)
418     """
419     if num1 < 0:
420         num1 = -num1
421     if num2 < 0:
422         num2 = -num2
423     shift = 0
424     # Если одно из чисел равно нулю, делитель - другое число

```

```

424     if num1 == 0:
425         return num2
426     if num2 == 0:
427         return num1
428     # Если num1 = 1010, а num2 = 0100, то num1 | num2 = 1110
429     # 1110 & 0001 == 0, тогда происходит сдвиг, который фиксируется в shift
430     while (num1 | num2) & 1 == 0:
431         shift += 1
432         num1 = num1 >> 1
433         num2 = num2 >> 1
434     # Если True, значит num1 - четное, иначе - нечетное
435     while num1 & 1 == 0:
436         # если нечетное, сдвигаем на один бит
437         num1 = num1 >> 1
438     while num2 != 0:
439         # пока число нечётное, сдвигаем на один бит
440         while num2 & 1 == 0:
441             num2 = num2 >> 1
442         # если первое число больше второго
443         if num1 > num2:
444             # меняем их местами
445             num1, num2 = num2, num1
446         # теперь первое число меньше второго, вычитаем
447         num2 = num2 - num1
448     # возвращаем число, перед этим сдвинув его биты на shift
449     return num1 << shift
450
451
452 if __name__ == '__main__':
453     while True:
454         menu_text = '\n'.join([
455             'Выберите действие:',
456             '1) x + y',
457             '2) x - y',
458             '3) x * y',
459             '4) x / y',
460             'q) Выход'
461         ])
462         print(menu_text)
463         choice = input()
464         if choice == '1':

```



```
465         x = BigInt(input('Введите первоечисло(x): '))
466         y = BigInt(input('Введите второечисло(y): '))
467         t = time.time()
468         res = x + y
469         print(f'\nВремя расчета: {time.time() - t} сек.')
470         print('x + y =', res, '\n\n')
471     elif choice == '2':
472         x = BigInt(input('Введите первоечисло(x): '))
473         y = BigInt(input('Введите второечисло(y): '))
474         t = time.time()
475         res = x - y
476         print(f'\nВремя расчета: {time.time() - t} сек.')
477         print('x - y =', res, '\n\n')
478     elif choice == '3':
479         x = BigInt(input('Введите первоечисло(x): '))
480         y = BigInt(input('Введите второечисло(y): '))
481         t = time.time()
482         res = x * y
483         print(f'\nВремя расчета: {time.time() - t} сек.')
484         print('x * y =', res, '\n\n')
485     elif choice == '4':
486         x = BigInt(input('Введите первоечисло(x): '))
487         y = BigInt(input('Введите второечисло(y): '))
488         t = time.time()
489         res = x / y
490         print(f'\nВремя расчета: {time.time() - t} сек.')
491         print('x / y =', res, '\n\n')
492     else:
493         exit()
```

Список литературы

1. *Акритас А.* Основы компьютерной алгебры с приложениями. — М. : МИР, 1994.
2. *Майрова С. П., Завгородний М. Г.* Программирование. Криптографические алгоритмы: учебное пособие. — Воронеж : Издательский дом ВГУ, 2018.
3. *Саммерфилд М.* Программирование на Python 3. Подробное руководство. — Символ-Плюс, 2009.