Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

За долгое время развития цифровой индустрии появилось множество методик по написанию программного обеспечения. Одной из самых популярных и удобных парадигм среди всех является объектно-ориентированное программирование. Программа, написанная в рамках этой парадигмы, представляет систему объектов различных типовб которые взаимодействуют между собой.

Такое восприятие программы помогает облегчить процесс разработки программы, так как почти любая функционирующая система в той или иной степени состоит из объектов, находящихся во взаимосвязи между собой.

Актуальность разработки курсовой работы K_3 "Моделирование работы инженерного калькулятора" обусловлена важностью знания о способах организации взаимодействия объектов вне системы взаимосвязи. Механизм сигналов и обработчиков реализует схему взаимодействия объектов один ко многим.

Цели данной курсовой работы: получить практические навыки разработки на языке программирования C++, создать программное обеспечение с функционалом работы инженерного арифметического калькулятора. При получении навыков разработки на C++ нужно усвоить материал основы работы

с классами.

Постановленной задачей является моделирование работы инженерного арифметического калькулятора на языке C++.

Для выполнения поставленной задачи нужно изучить теоретическую часть создания инженерного арифметического калькулятора. Для создания функционала управления понадобится знание в правильной аллокации памяти и реализации взаимодействия объектов по средствам сигналов и обработчиков.

1 ПОСТАНОВКА ЗАДАЧИ

Надо моделировать работу калькулятора следующей конструкции:

- в вычислении участвуют целые числа объемом памяти 2 байта;
- допустимые операции: +, -, *, / (целочисленное деление), % (деление с остатком), << (побитовый сдвиг влево), >> (побитовый сдвиг в право);
- операции выполняются последовательно, для выполнения операции необходимы два аргумента и знак операции;
- после выполнения каждой операции фиксируется и выводится результат;
- последовательность операций и аргументов образует выражение;
- результат отображается в 16, 10 и 2-ой системе счисления;
- при возникновении переполнения выдается Overflow;
- при попытке деления на 0 выдается Division by zero;
- при вводе знака “C” калькулятор приводиться в исходное состояние, первый аргумент выражения принимает значение 0 и готов для ввода очередного выражения;
- при вводе знака “Off” калькулятор завершает работу.

Нажатие на клавиши калькулятора моделируется посредством клавиатурного ввода. Ввод делится на команды:

- «целое число» - первый аргумент выражения, целое не отрицательное число, можно последовательно вводить несколько раз, предыдущее значение меняется. При вводе не первым аргументом выражения - игнорируется;
- «знак операции» «целое число» - второе и последующие операции выражения;
- «C» - приведение калькулятора в исходное состояние;
- «Off» - завершение работы калькулятора.

Вывод результата моделируется посредством вывода на консоли. Результат

выводиться в следующей форме:

```
«выражение»     HEX «16-ое число»  DEC «10-ое число»  BIN «2-ое число»
```

«16-ое число» выводиться в верхнем регистре с лидирующими нулями (пример 01FA).

«10-ое число» (пример 1765).

«2-ое число» выводиться разбивкой по четыре цифры с лидирующими нулями (пример 0000 0100 0111 0101).

Построить систему, которая использует объекты:

1. Объект «система».
2. Объект для чтения команд. После чтения очередной команды объект выдает сигнал с текстом, содержащим команду. Все команды синтаксический корректны (моделирует пульт управления калькулятора).
3. Объект для выполнения арифметических операции. После завершения выдается сигнал с текстом результата. Если произошло переполнение или деление на нуль, выдается сигнал об ошибке. После выдачи сообщения калькулятор переводится посредством соответствующего сигнала в исходное положение.
4. Объект для выполнения операции побитового сдвига. После завершения выдается сигнал с текстом результата.
5. Объект для выполнения операции «C».
6. Объект для вывода очередного результата на консоль.

Написать программу, реализующую следующий алгоритм:

1. Вызов метода объекта «система» build_tree_objects ( ).
    1.1. Построение дерева иерархии объектов.
    1.2. Установка связей сигналов и обработчиков между объектами.
2. Вызов метода объекта «система» exec_app ( ).
    2.1. Приведение всех объектов в состояние готовности.

2.2. Цикл для обработки вводимых команд.

2.2.1. Выдача сигнала объекту для ввода команды.

2.2.2. Отработка команды.

2.3. После ввода команды «Off» завершить работу.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы.

Все сообщения на консоль выводятся с новой строки.

В набор поддерживаемых команд добавить команду «SHOWTREE» и по этой команде вывести дерево иерархии объектов системы с отметкой о готовности и завершить работу программы.

1.1 Описание входных данных

Построчно множество команд, в любом количестве. Перечень команд:

```
«целое не отрицательное число»
«знак операции» «целое число»
C
```

Последняя команда присутствует всегда:

```
Off
```

Пример ввода:

```
5
+ 5
<< 1
/ 0
+ 5
C
7
8
/ -3
C
9
% -4
+ 7
* 11
Off
```

1.2 Описание выходных данных

Построчно выводиться результат каждой операции по форме:

```
«выражение»     HEX «16-ое число»  DEC «10-ое число»  BIN «2-ое число»
```

Если произошло переполнение:

```
«выражение»     Overflow
```

Если произошло переполнение:

```
«выражение»     Division by zero
```

Пример вывода:

```
5 + 5     HEX 000A  DEC 10  BIN 0000 0000 0000 1010
5 + 5 << 1     HEX 0014  DEC 20  BIN 0000 0000 0001 0100
5 + 5 << 1 / 0     Division by zero
0 + 5     HEX 0005  DEC 5  BIN 0000 0000 0000 0101
8 / -3     HEX FFFE  DEC -2  BIN 1111 1111 1111 1110
9 % -4     HEX 0001  DEC 1  BIN 0000 0000 0000 0001
9 % -4 + 7     HEX 0008  DEC 8  BIN 0000 0000 0000 1000
9 % -4 + 7 * 11     HEX 0058  DEC 88  BIN 0000 0000 0101 1000
```

2 МЕТОД РЕШЕНИЯ

Класс Base:

- функционал:
    - метод Base — конструктор;
    - метод ~Base — деструктор;
    - метод print — выводит иерархию объектов с отметками о готовности;
    - метод is_ready — возвращает готовность объекта;
    - метод set_children_state — станавливает состояние подчинённых объектов;
    - метод add_signal_handler — добавляет обработчик сигнала;
    - метод remove_signal_handler — удаляет обработчик сигнала;
    - метод emit_signal — запускает обработчики сигнала.

Класс connections:

- свойства/поля:
    - поле объект:
        - наименование — object;
        - тип — указатель на Base;
        - модификатор доступа — public;
    - поле обработчик:
        - наименование — handler;
        - тип — указатель на метод-обработчик класса Base;
        - модификатор доступа — public;

Класс ArithmeticOperations:

- функционал:
    - метод ArithmeticOperations — конструктор;

- метод math — выполнение операции.

Класс BitwiseOperations:

- функционал:
  - метод BitwiseOperations — конструктор;
  - метод math — выполнение операции.

Класс Input:

- функционал:
  - метод Input — конструктор;
  - метод read_line — чтение строки.

Класс Output:

- функционал:
  - метод Output — конструктор;
  - метод print — вывод результата операции.

Класс Reset:

- функционал:
  - метод Reset — конструктор.

Класс System:

- функционал:
  - метод System — конструктор;
  - метод build_tree_objects — построение иерархии объектов;
  - метод exec_app — запуск приложения.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Base				
2	connections				

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
3	ArithmeticOperations				
4	BitwiseOperations				
5	Input				
6	Output				
7	Reset				
8	System				
		ArithmeticOperations	private		3
		BitwiseOperations	private		4
		Input	private		5
		Output	private		6

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм конструктора класса Base

Функционал: конструктор.

Параметры: head_object (указатель на Base), name ().

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса Base

№	Предикат	Действия	№ перехода
1		присваивание state значение 0	2
2		присваивание this.name значение name	3
3		присваивание this.head_object значение head_object	4
4	head_object не равно 0	добавление this в head_object.subordinate_objects	∅
			∅

3.2 Алгоритм деструктора класса Base

Функционал: деструктор.

Параметры: void.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса Base

№	Предикат	Действия	№ перехода
1		удаление каждого объекта из subordinate_objects	∅

3.3 Алгоритм метода print класса Base

Функционал: выводит иерархию объектов с отметками о готовности.

Параметры: int level.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода print класса Base

№	Предикат	Действия	№ перехода
1		вызов метода print_level_padding() с параметром level	2
2		вывод name, отметки о готовности объекта	3
3		для каждого подчиненного объекта object вызов метода print() с параметром level+1	∅

3.4 Алгоритм метода is_ready класса Base

Функционал: возвращает готовность объекта.

Параметры: int.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода is_ready класса Base

№	Предикат	Действия	№ перехода
1		возврат state!=0 и (head_object=0 или head_object.is_ready())	∅

3.5 Алгоритм метода set_children_state класса Base

Функционал: станавливает состояние подчинённых объектов.

Параметры: int state.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_children_state класса Base

№	Предикат	Действия	№ перехода
1		присваивание this.state state	2
2		для каждого подчиненного объекта вызов метода set_children_state() с параметром state	∅

3.6 Алгоритм метода add_signal_handler класса Base

Функционал: добавляет обработчик сигнала.

Параметры: object (указатель на класс Base), handler (обработчик).

Возвращаемое значение: none.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода add_signal_handler класса Base

№	Предикат	Действия	№ перехода
1	handlers содержит этот обработчик		∅
		добавление обработчика в handlers	∅

3.7 Алгоритм метода remove_signal_handler класса Base

Функционал: удаляет обработчик сигнала.

Параметры: object (указатель на класс Base), handler (обработчик).

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода remove_signal_handler класса Base

№	Предикат	Действия	№ перехода
1		удаление обработчика из handlers	∅

3.8 Алгоритм метода emit_signal класса Base

Функционал: запускает обработчики сигнала.

Параметры: string message.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода emit_signal класса Base

№	Предикат	Действия	№ перехода
1	!is_ready		∅
		вызов обработчика каждого объекта с аргументом message	∅

3.9 Алгоритм функции main

Функционал: главная функция программы.

Параметры: none.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 10.

Таблица 10 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта app класса System	2
2		вызов метода build_tree_objects() объекта app	3
3		возврат результата выполнения метода exec_app объекта app	∅

3.10 Алгоритм конструктора класса ArithmeticOperations

Функционал: конструктор.

Параметры: head_object (указатель на Base).

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса ArithmeticOperations

№	Предикат	Действия	№ перехода
1		вызов конструктора Base с аргументами head)object, "ArithmeticOperations"	∅

3.11 Алгоритм метода math класса ArithmeticOperations

Функционал: выполнение операции.

Параметры: двухбайтовое целое a, int b, result (указатель на двухбайтовое целое), string operation, operations(ссылка на строку).

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода math класса ArithmeticOperations

№	Предикат	Действия	№ перехода
1		присваивание operations operations + operation = " "+ результат вывода метода to_string() с параметром b+" "	2
2	b>32767 или b<-32768	вызов метода emit_signal() с параметром "Overflow"	3
			4
3		возврат "Overflow"	∅
4	operation="+"	присваивание result значение a+b	5
			7

№	Предикат	Действия	№ перехода
5	произошло переполнение	вызов метода emit_signal() с параметром "Overflow"	6
			19
6		возврат "Overflow"	∅
7	operation="-"	присваивание result значение a-b	8
			10
8	произошло переполнение	вызов метода emit_signal() с параметром "Overflow"	9
			19
9		возврат "Overflow"	∅
10	operation="*"	присваивание result значение a*b	11
			13
11	произошло переполнение	вызов метода emit_signal() с параметром "Overflow"	12
			19
12		возврат "Overflow"	∅
13	operation="/"		14
			16
14	b=0	вызов метода emit_signal() с параметром "Division by zero"	15
		присваивание result значение a/b	19
15		возврат "Division by zero"	∅
16	operation="%"		17
			19
17	b=0	вызов метода emit_signal() с параметром "Division by zero"	18
		присваивание result значение a%b	19
18		возврат "Division by zero"	∅
19		вызов метода emit_signal с параметром (результат	20

№	Предикат	Действия	№ перехода
		вызова метода to_string с параметром result)	
20		возврат ""	∅

3.12 Алгоритм конструктора класса BitwiseOperations

Функционал: конструктор.

Параметры: head_object (указатель на класс Base).

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса BitwiseOperations

№	Предикат	Действия	№ перехода
1		вызов конструктора Base с аргументами head_object, "BitwiseOperations"	∅

3.13 Алгоритм метода math класса BitwiseOperations

Функционал: выполнение операции.

Параметры: двухбайтовое целое a, int b, result (указатель на двухбайтовое целое), string operation, operations(ссылка на строку).

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода math класса BitwiseOperations

№	Предикат	Действия	№ перехода
1		присваивание operations operations + operation = " "+ результат вывода метода to_string() с параметром b+" "	2
2	b>32767 или b<-32768	вызов метода emit_signal() с параметром "Overflow"	3

№	Предикат	Действия	№ перехода
			4
3		возврат "Overflow"	∅
4	operation="<<"	присваивание result значение a<<b	6
			5
5	operation=">>"	присваивание result значение a>>b	6
			6
6		вызов метода emit_signal с параметром (результат вызова метода to_string с параметром result)	7
7		возврат ""	∅

3.14 Алгоритм конструктора класса Input

Функционал: конструктор.

Параметры: head_object (указатель на Base).

Алгоритм конструктора представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса Input

№	Предикат	Действия	№ перехода
1		вызов конструктора с параметрами head_object, "Input"	∅

3.15 Алгоритм метода read_line класса Input

Функционал: чтение строки.

Параметры: none.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода read_line класса Input

№	Предикат	Действия	№ перехода
1		объявление переменной input типа string	2
2		чтение input	3
3	размер input не равен 0 и input[размер input] равен '\n'	удаление у input последнего элемента	4
			4
4	размер input не равен 0 и input[размер input] равен '\r'	удаление у input последнего элемента	5
			5
5		вызов метода emit_signal с параметром input	6
6		возврат input	∅

3.16 Алгоритм конструктора класса Output

Функционал: конструктор.

Параметры: head_object(указатель на Base).

Алгоритм конструктора представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса Output

№	Предикат	Действия	№ перехода
1		вызов конструктора с параметрами head_object, "Output"	∅

3.17 Алгоритм метода print класса Output

Функционал: вывод результата операции.

Параметры: operations (строка), result (двухбайтовое целое), last_operation_result (строка).

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода print класса Output

№	Предикат	Действия	№ перехода
1		удаление последнего элемента у operations	2
2		вывод operations, " "	3
3	last_operation_Result не равно ""	вывод last_operation_result	∅
		вывод "HEX ", result, " DEC ", result, " BIN"	4
4		объявление строковой переменной bin равной bitset<16>(result).to_string()	5
5		объявление переменной i типа int со значением 0	6
6	i=16	вывод "\n"	∅
			7
7	i%4=0	вывод " "	8
			8
8		вывод bin[i]	9
9		переменной i присваевается значение i+1	6

3.18 Алгоритм конструктора класса Reset

Функционал: конструктор.

Параметры: head_object (указатель на Base).

Алгоритм конструктора представлен в таблице 19.

Таблица 19 – Алгоритм конструктора класса Reset

№	Предикат	Действия	№ перехода
1		вызов конструктора с параметрами head_object, "Reset"	∅

3.19 Алгоритм конструктора класса System

Функционал: конструктор.

Параметры: none.

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – Алгоритм конструктора класса System

№	Предикат	Действия	№ перехода
1		вызов конструктора с параметрами 0, "System"	∅

3.20 Алгоритм метода build_tree_objects класса System

Функционал: построение иерархии объектов.

Параметры: none.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода build_tree_objects класса System

№	Предикат	Действия	№ перехода
1		создание объекта Input input с параметром this	2
2		создание объекта ArithmeticOperations arithmetic_operations с параметром this	3
3		создание объекта Output output с параметром this	4
4		создание объекта BitwiseOperations bitwise_operations с аргументом this	∅

3.21 Алгоритм метода exec_app класса System

Функционал: запуск приложения.

Параметры: none.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода exec_app класса System

№	Предикат	Действия	№ перехода
1		вызов метода set_children_state() с параметром 1	2
2		объявление переменных input, operations типа string	3
3		объявление целой двухбайтовой переменной result	4
4		присваивание input значение this.input.read_line()	5
5	input=="Off"	возврат значения 0	∅
			6
6	input=="SHOWTREE"	вызов метода print()	4
			7
7	input=="C"	вызов метода emit_signal с параметром C	4
			8
8	input[0] - цифра	присваивание result значение метода stoi с параметром input	9
			10
9		присваивание operation значение input +" "	4
10		объявление переменных operation, math_result типа string	11
11		объявление переменной b типа int	12
12		чтение operation, b из input	13
13	operation=">>" или operation="<<"	присвоение math результат метода bitwise_operations.math с параметрами result, b, result, operation, operations	14
		присвоение math результат метода arithmetic_operations.math с параметрами result, b, result, operation, operations	14
14		вызов метода output.print с параметрами (operations, result, math_result)	15
15	math_result не равен ""	присваивание result значение 0	16

№	Предикат	Действия	№ перехода
			4
16		присваивание operations значение метода to_string с параметром result +" "	4

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-24.

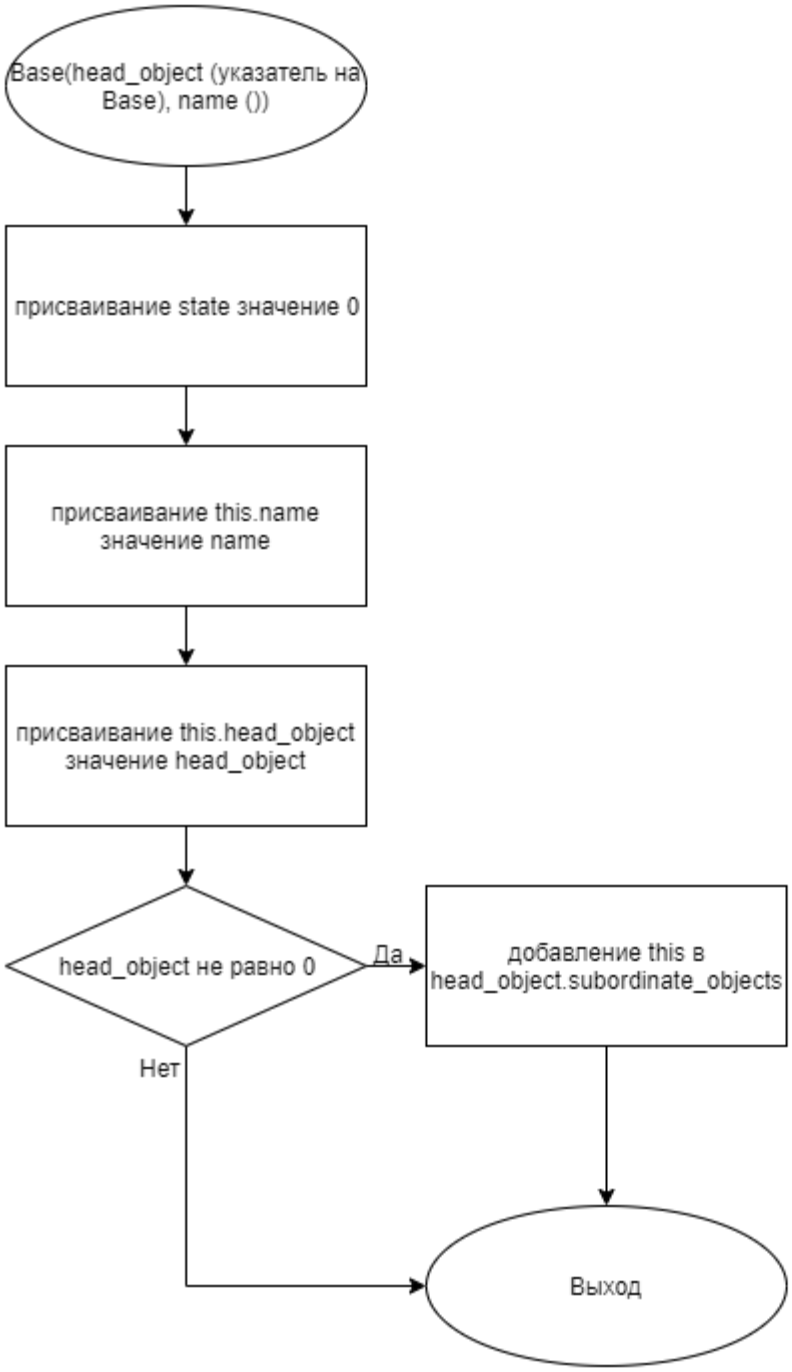

Рисунок 1 – Блок-схема алгоритма

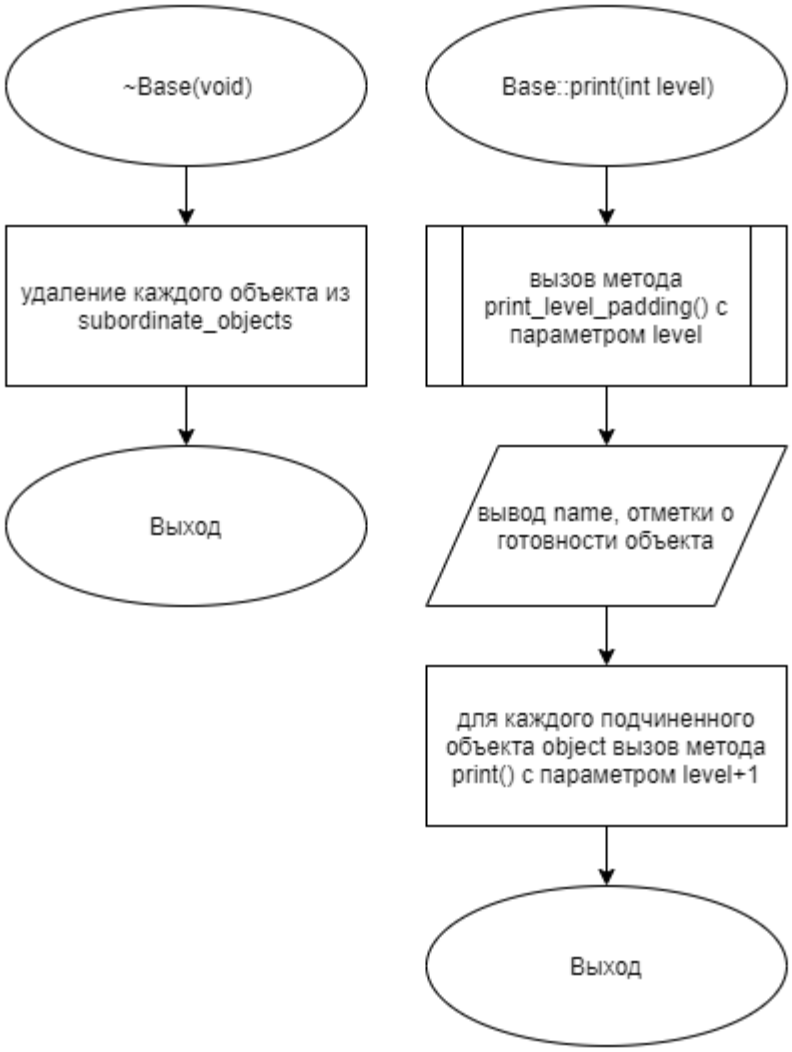

Рисунок 2 – Блок-схема алгоритма

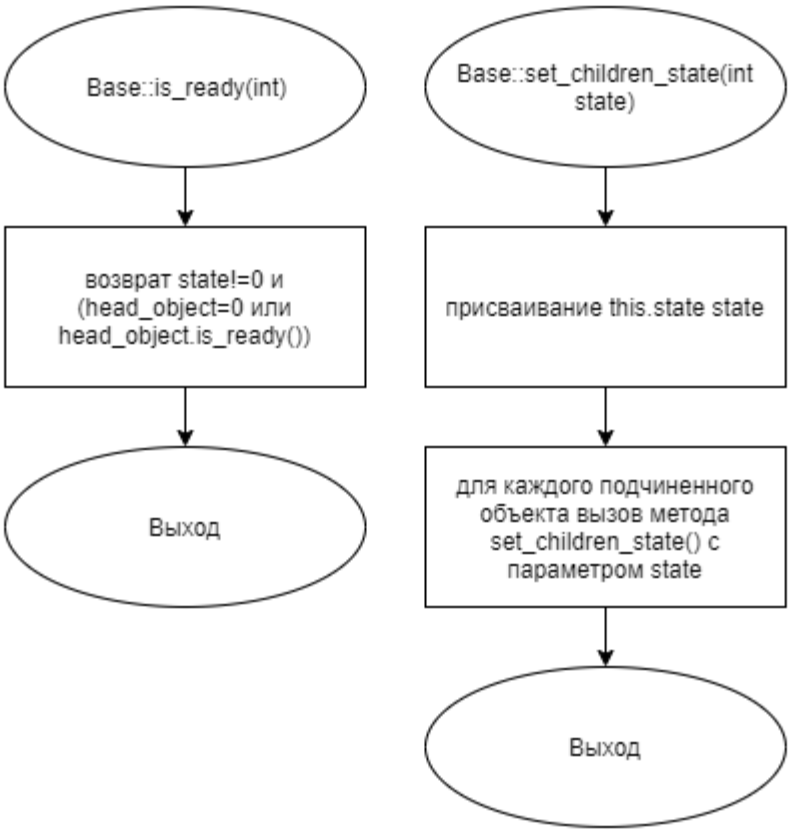

Рисунок 3 – Блок-схема алгоритма

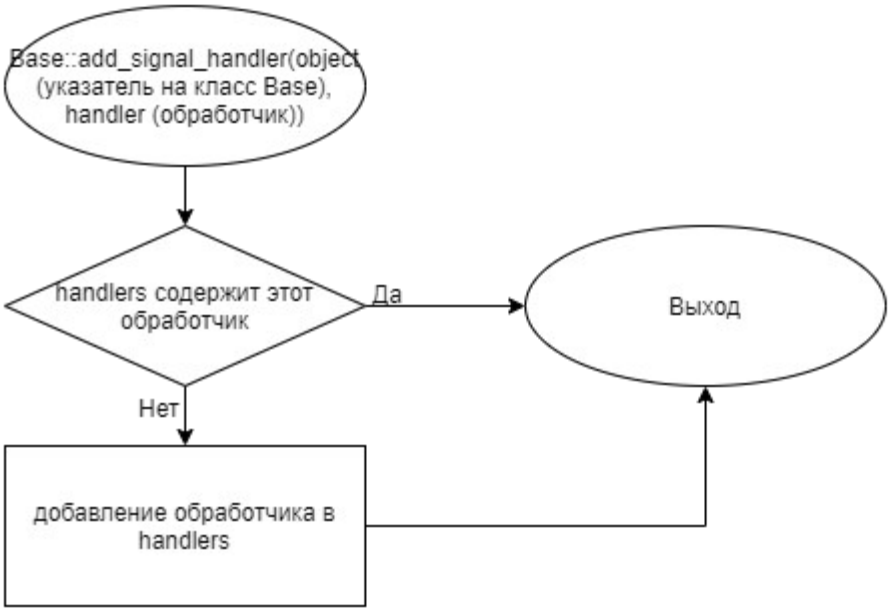

Рисунок 4 – Блок-схема алгоритма

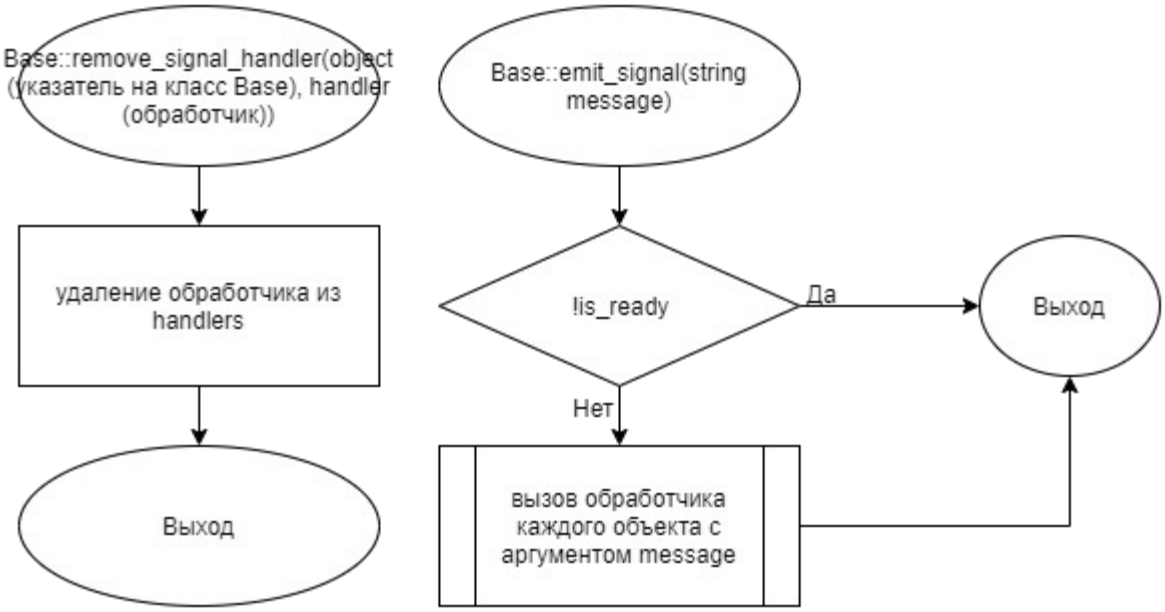

Рисунок 5 – Блок-схема алгоритма

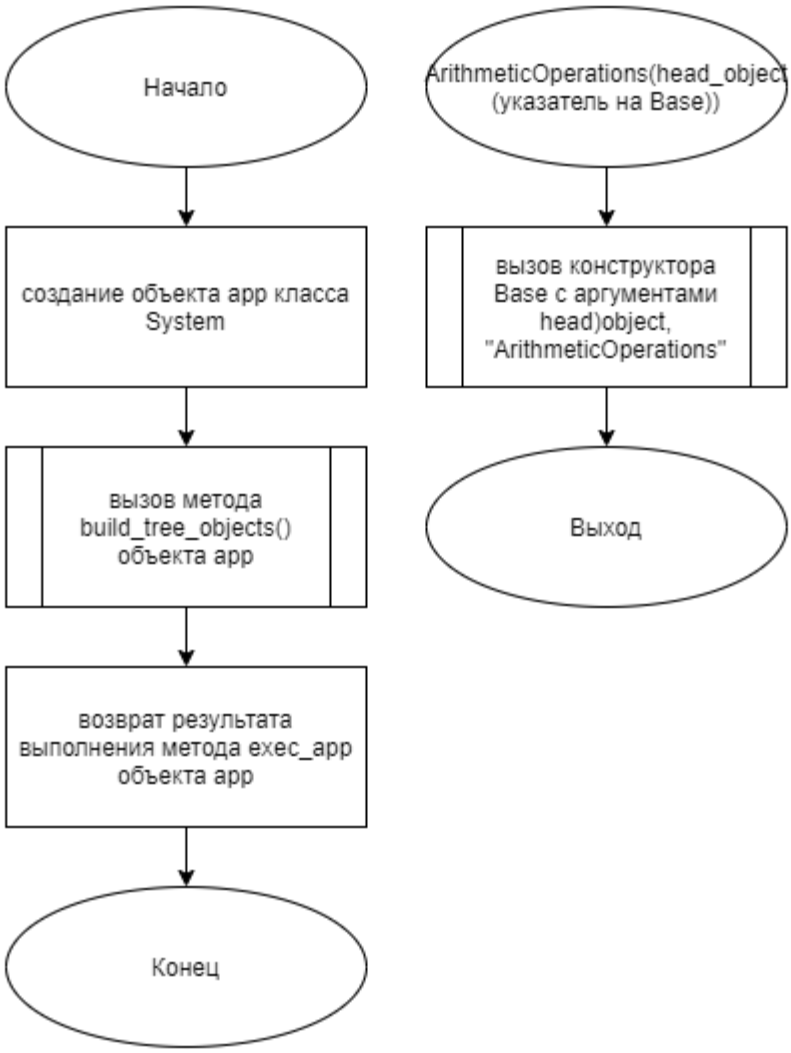

Рисунок 6 – Блок-схема алгоритма

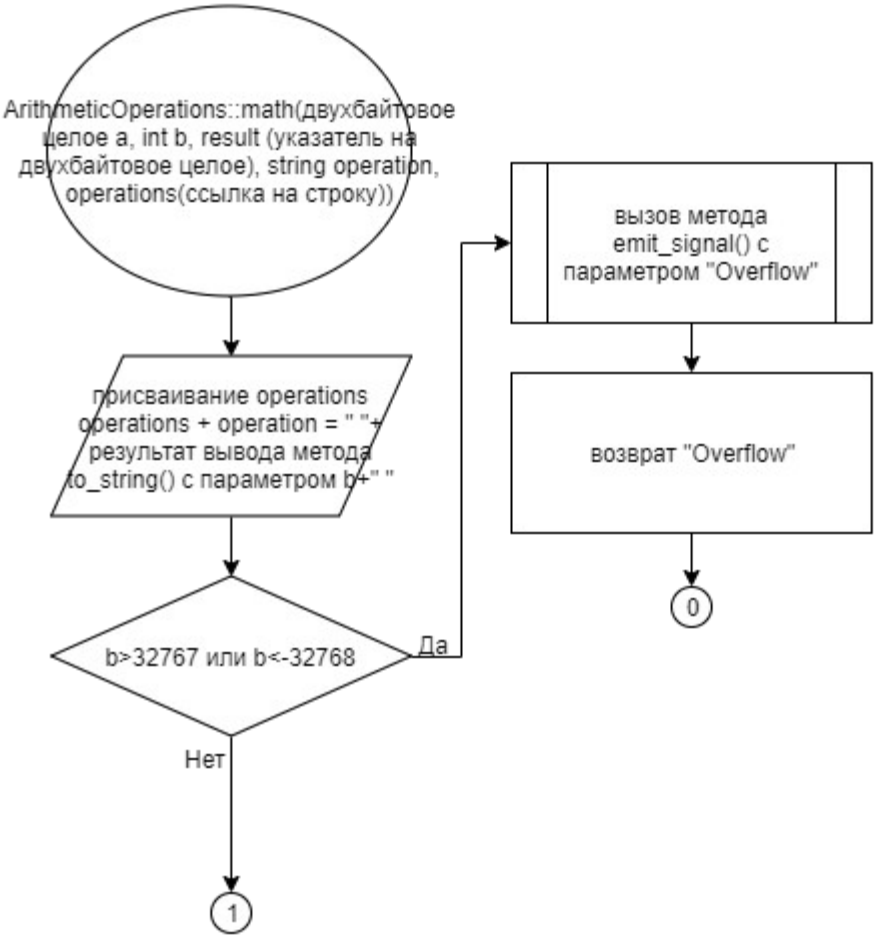

Рисунок 7 – Блок-схема алгоритма

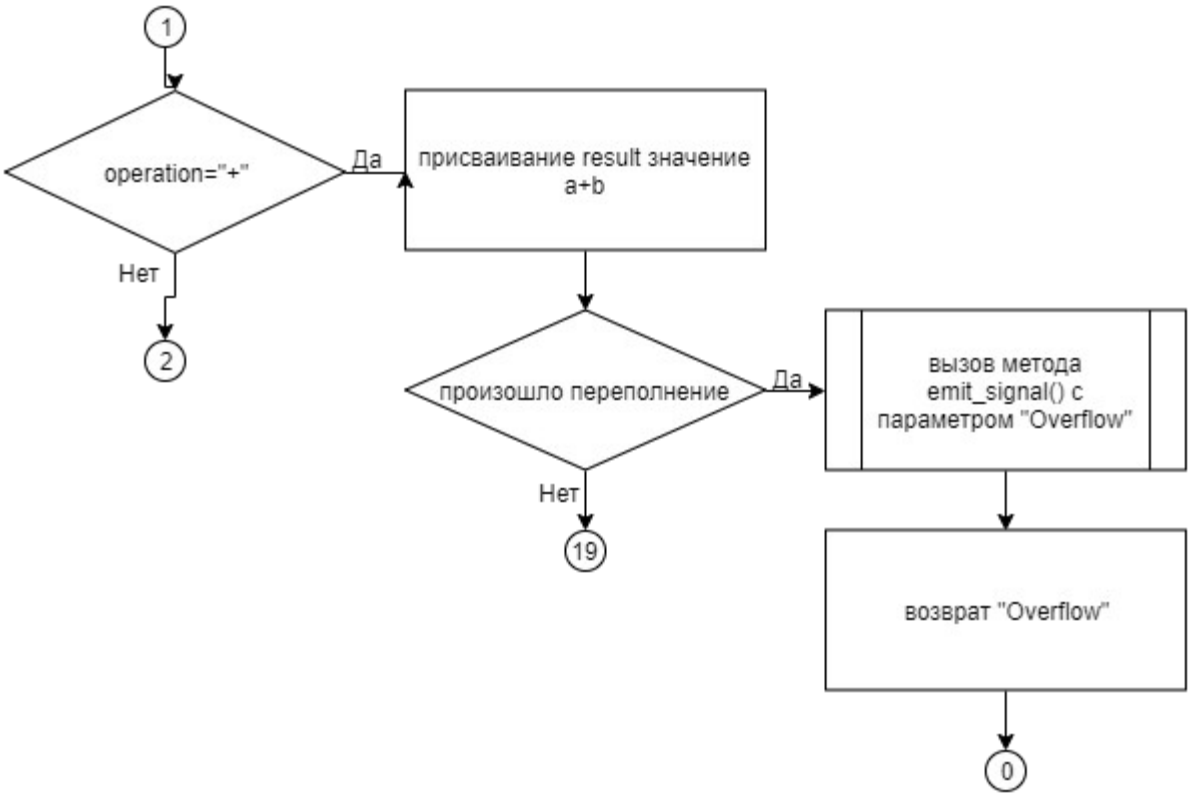

Рисунок 8 – Блок-схема алгоритма

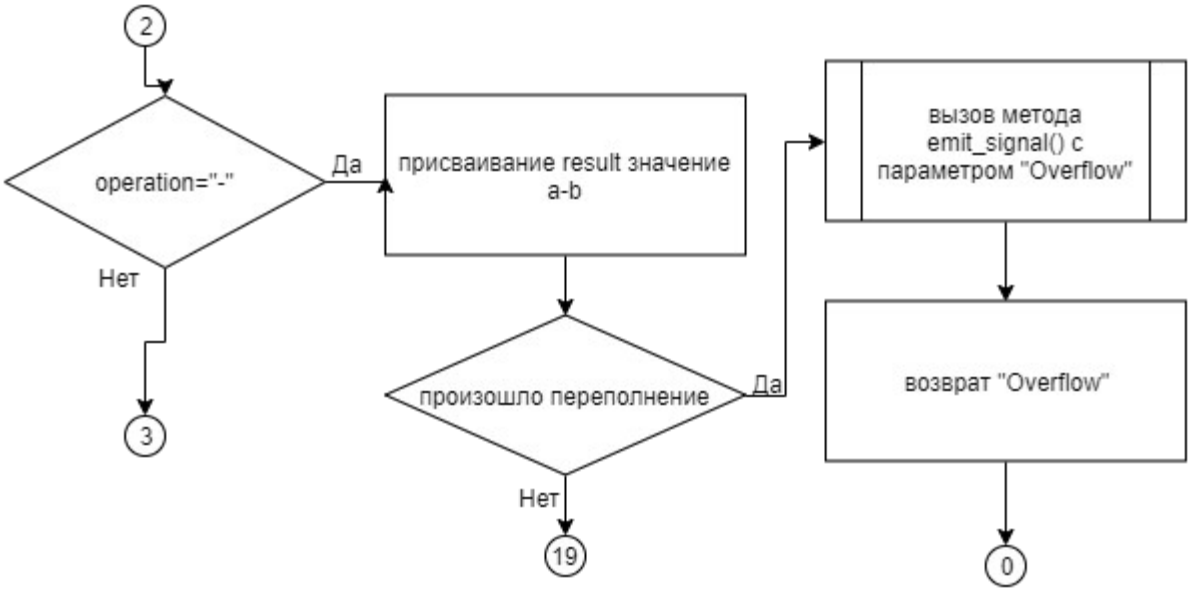

Рисунок 9 – Блок-схема алгоритма

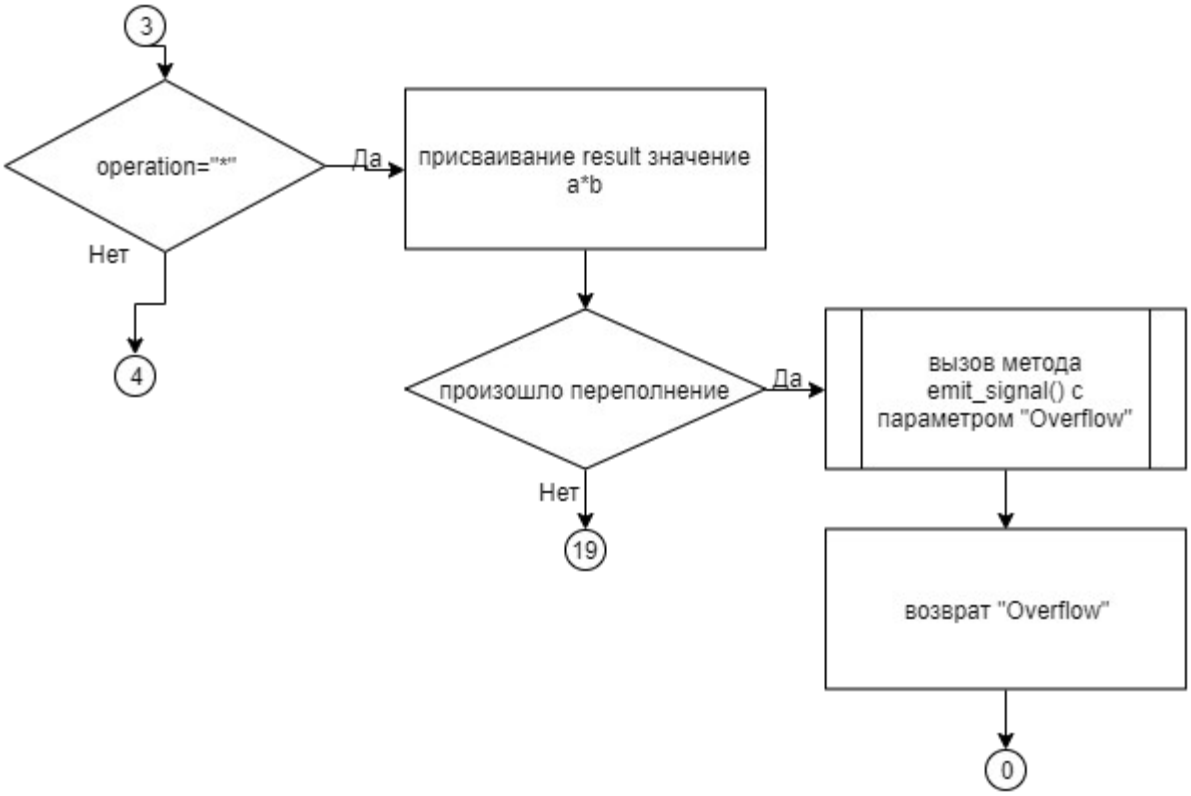

Рисунок 10 – Блок-схема алгоритма

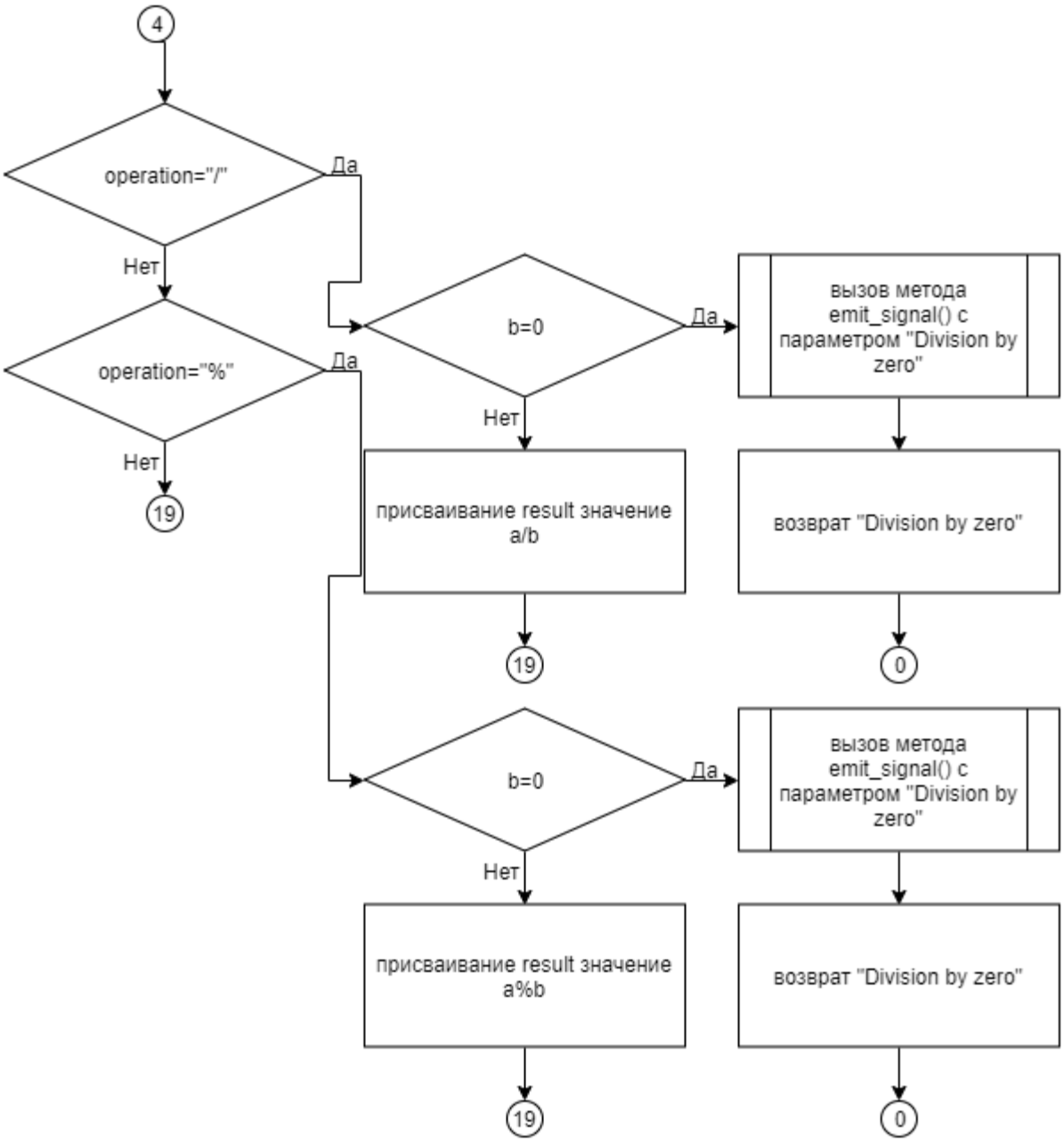

Рисунок 11 – Блок-схема алгоритма

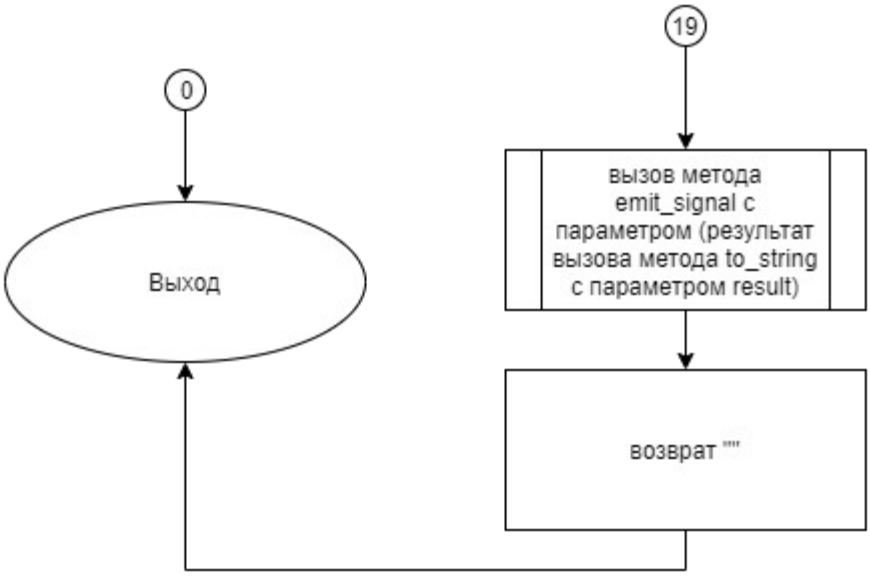

Рисунок 12 – Блок-схема алгоритма

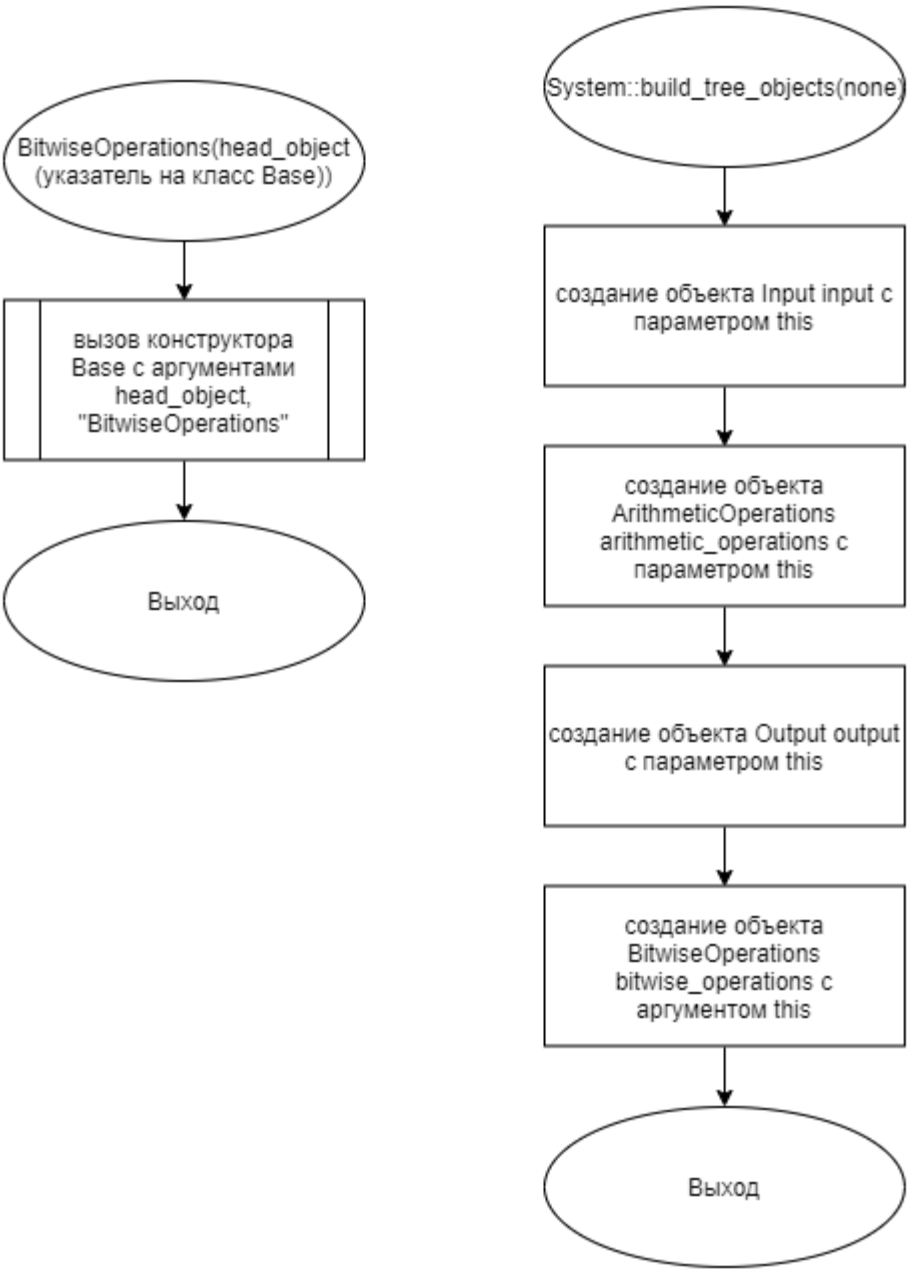

Рисунок 13 – Блок-схема алгоритма

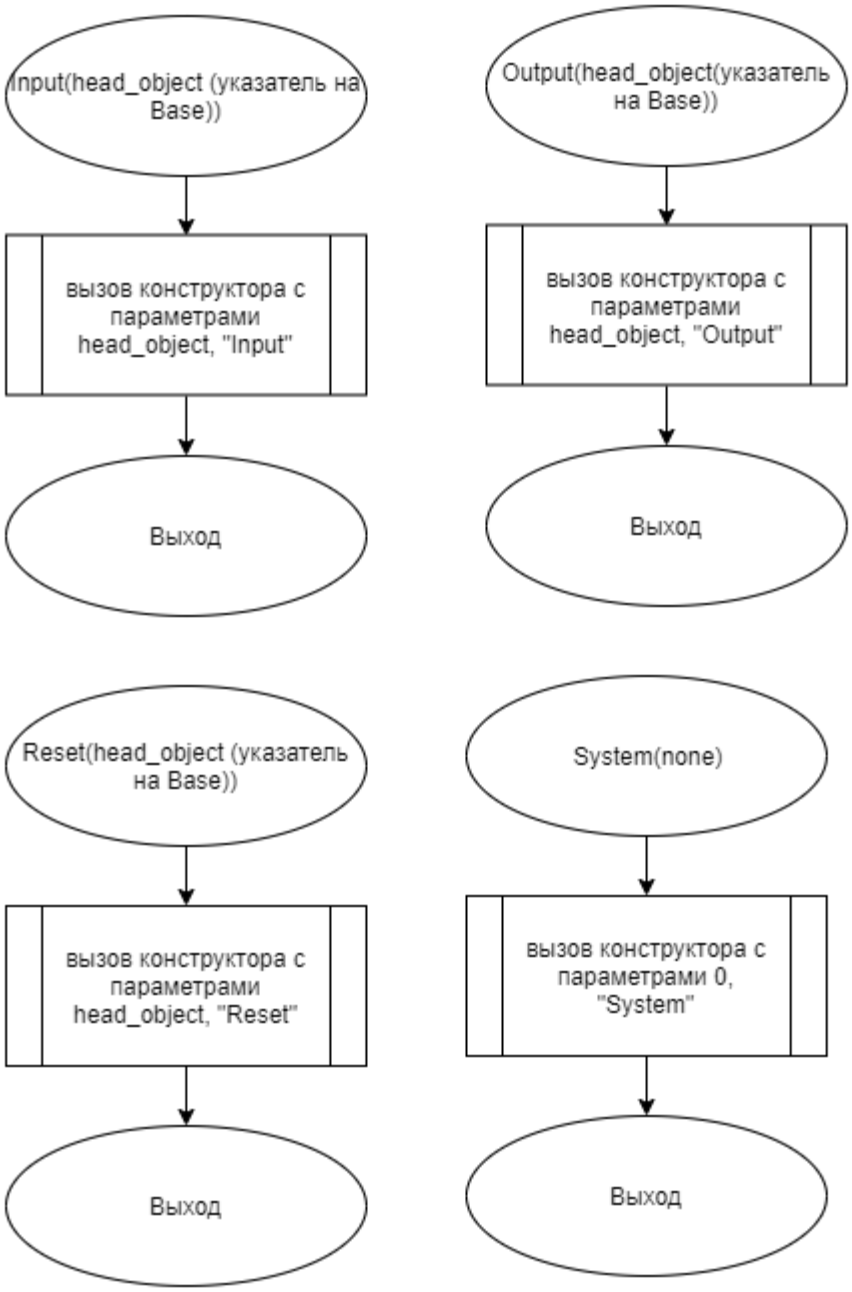

Рисунок 14 – Блок-схема алгоритма

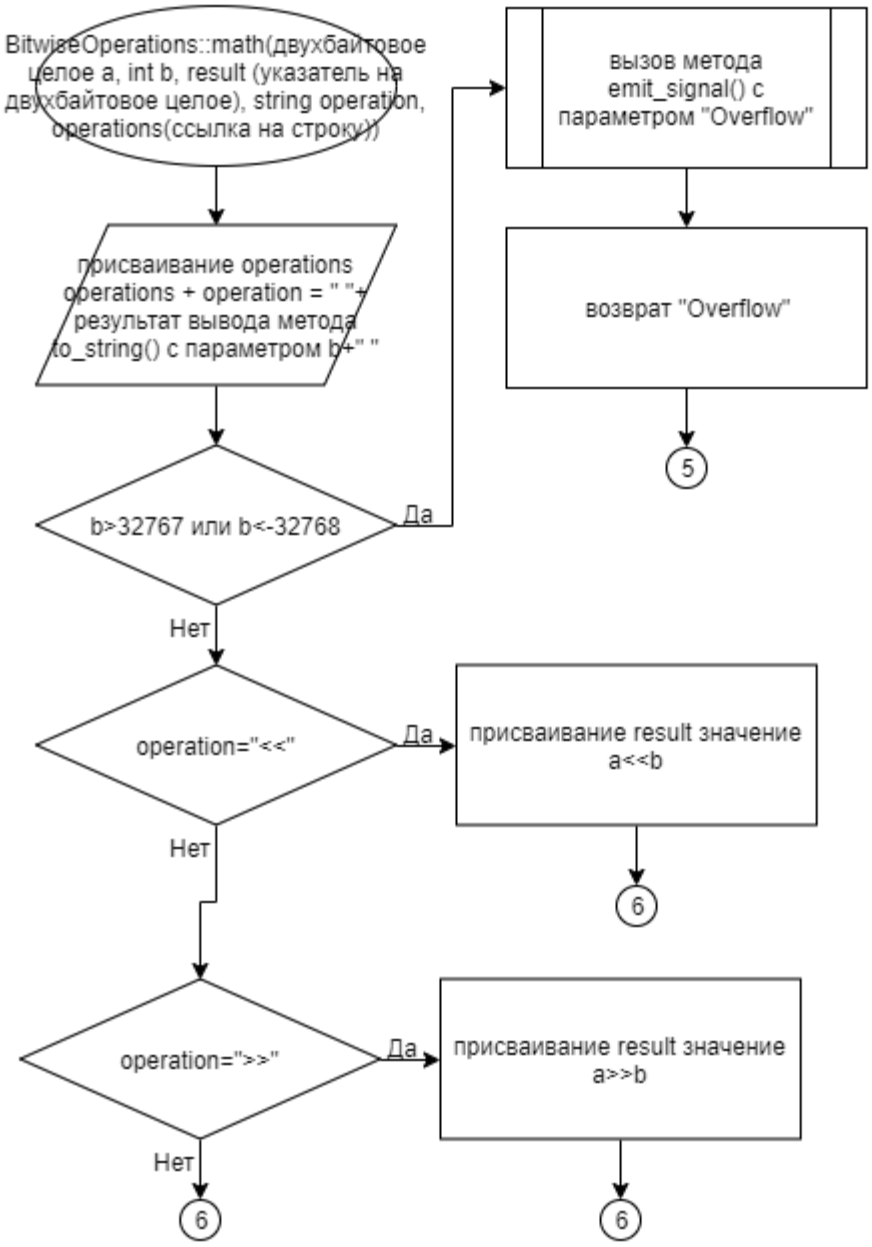

Рисунок 15 – Блок-схема алгоритма

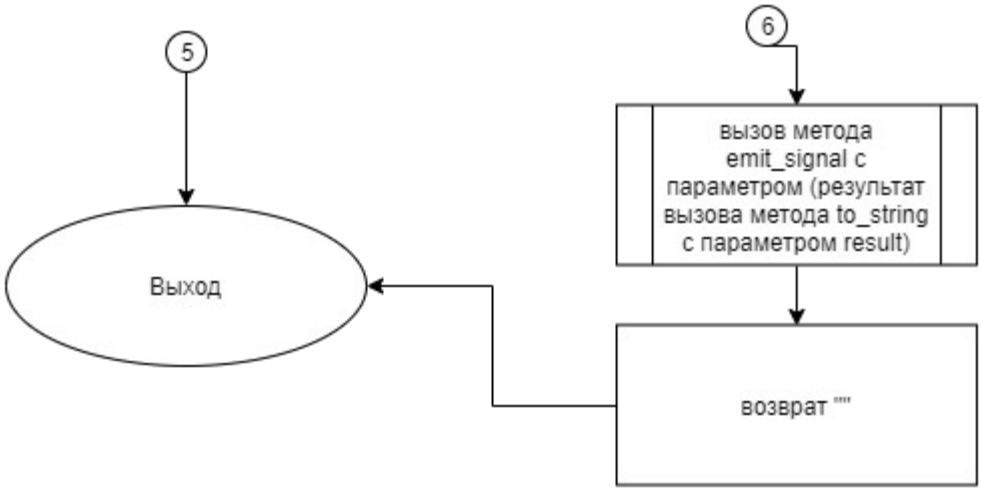

Рисунок 16 – Блок-схема алгоритма

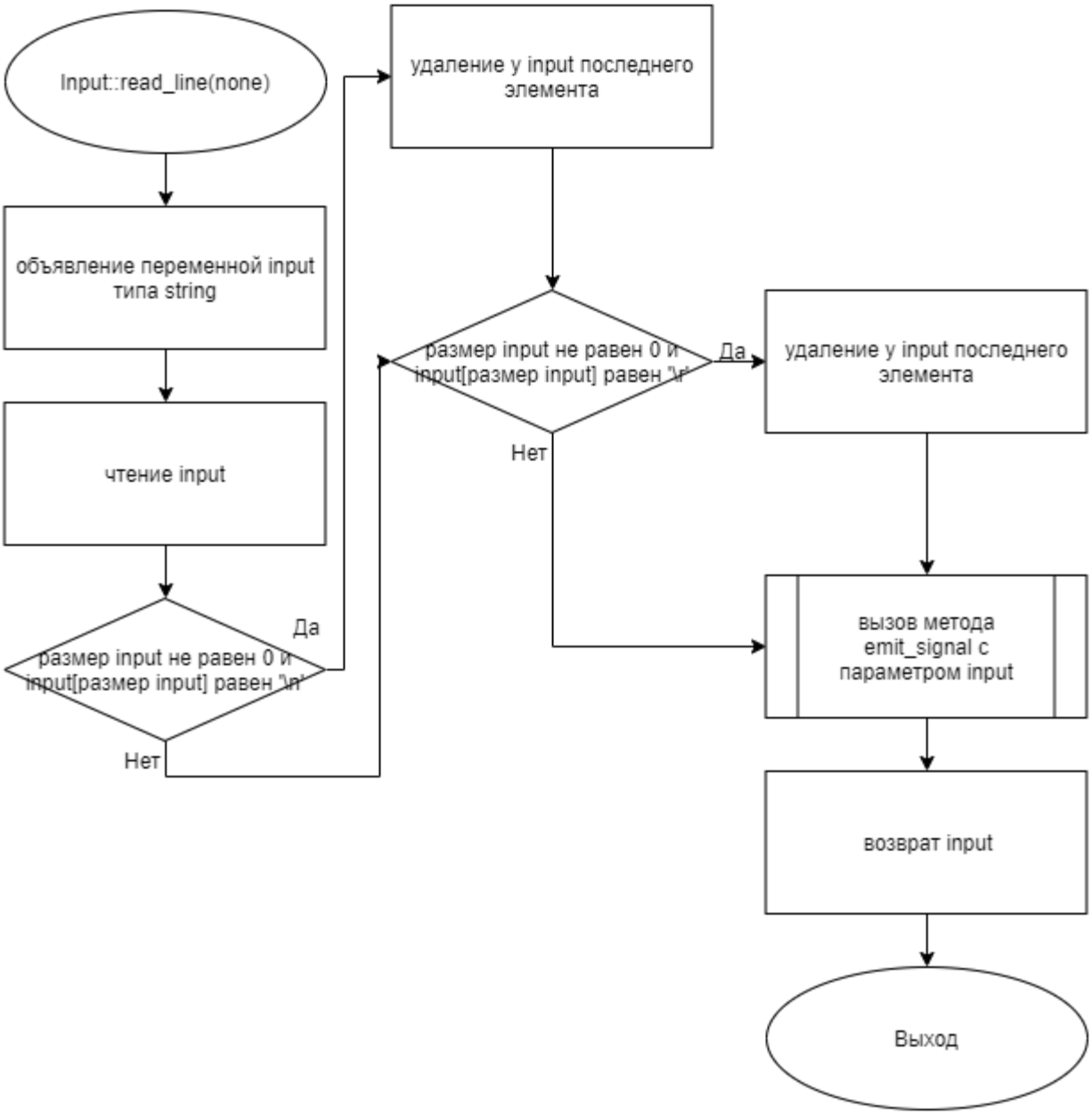

Рисунок 17 – Блок-схема алгоритма

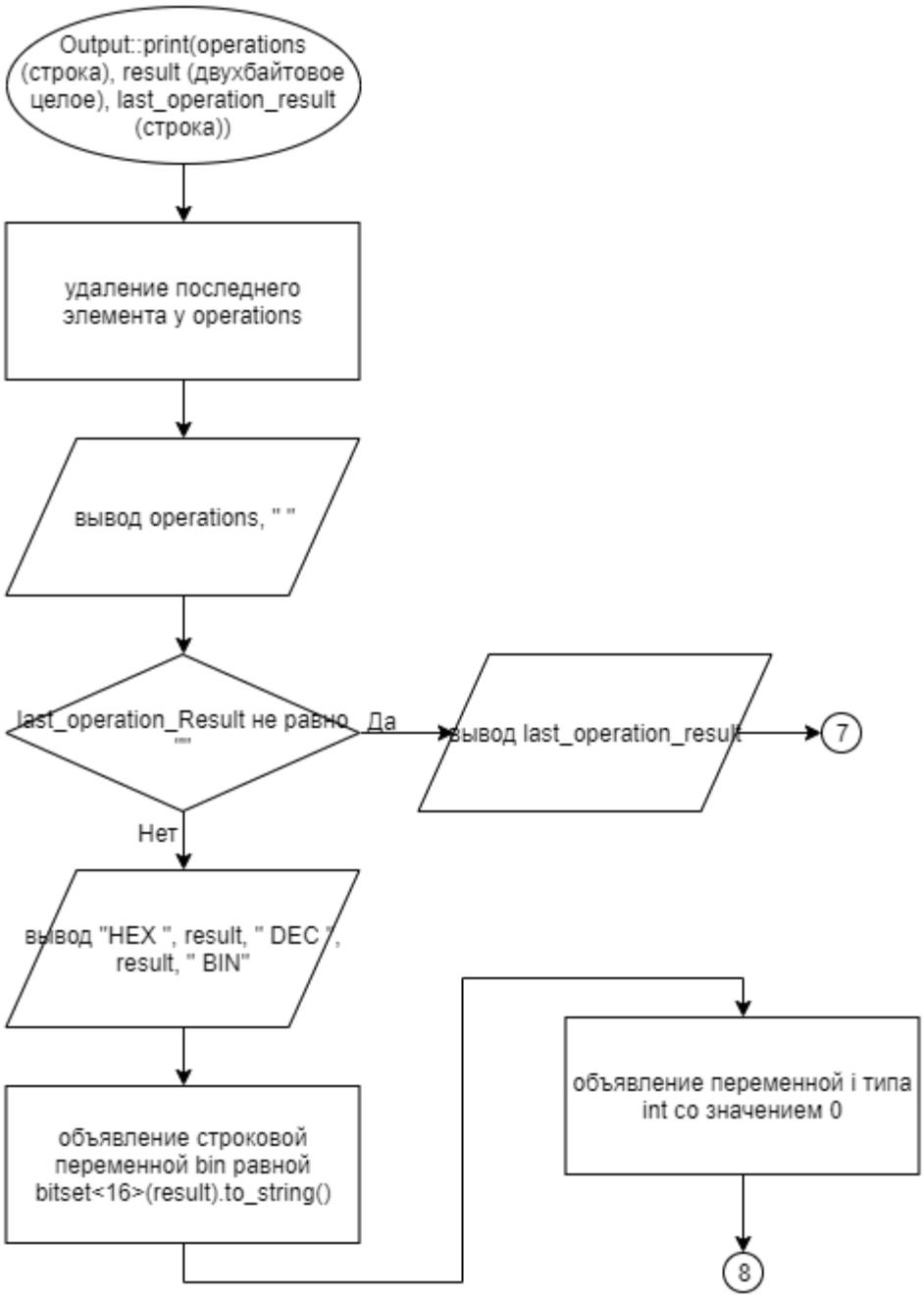

Рисунок 18 – Блок-схема алгоритма

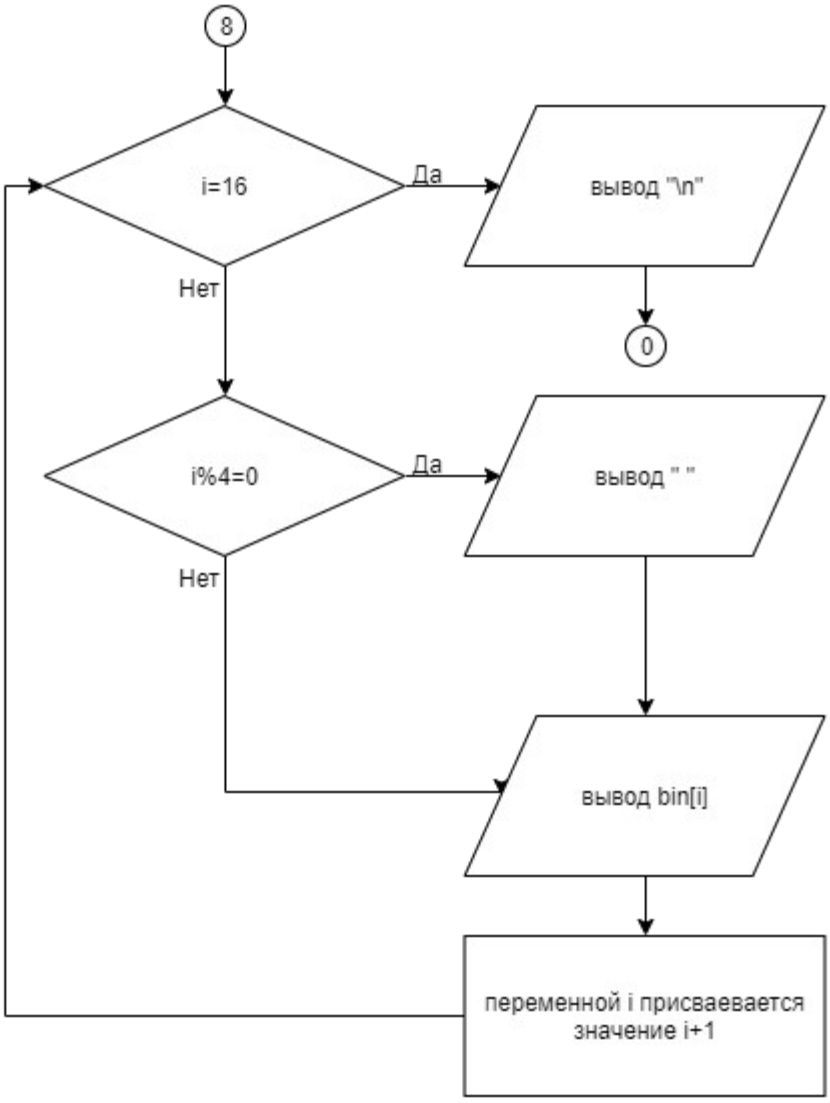

Рисунок 19 – Блок-схема алгоритма

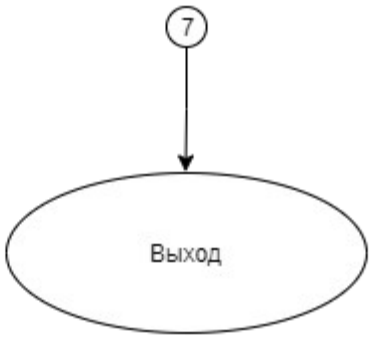

Рисунок 20 – Блок-схема алгоритма

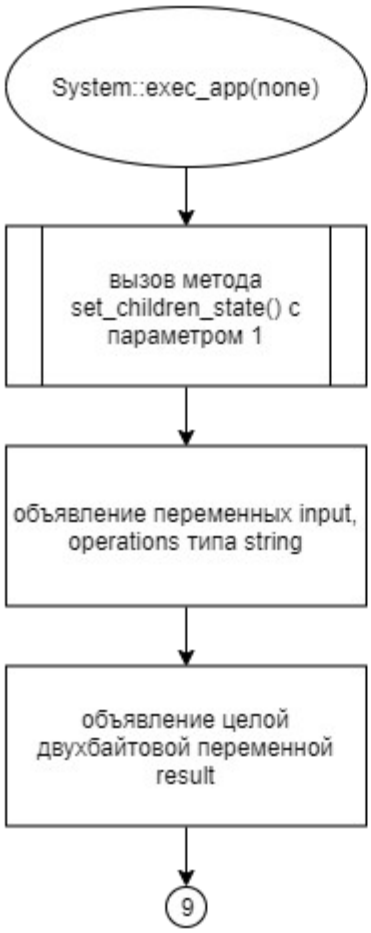

Рисунок 21 – Блок-схема алгоритма

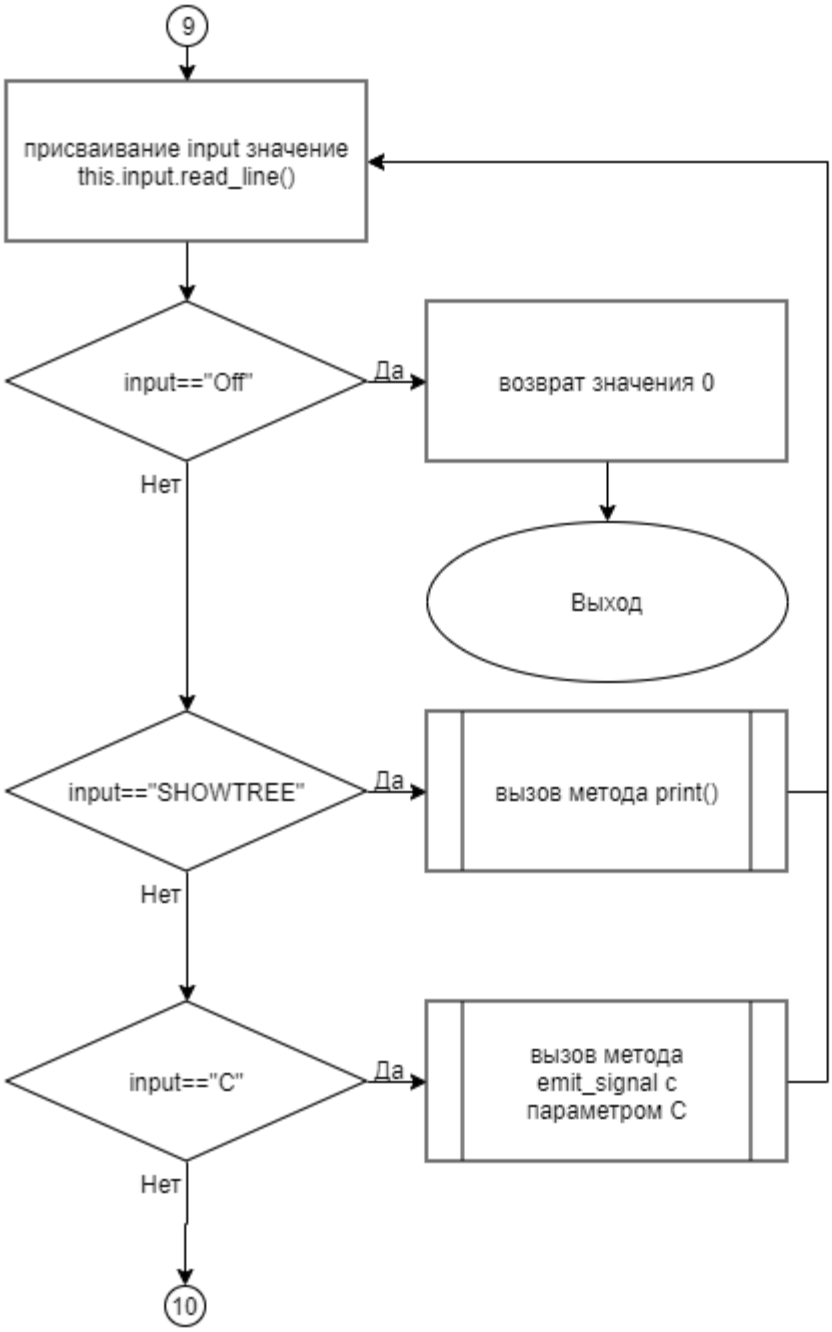

Рисунок 22 – Блок-схема алгоритма

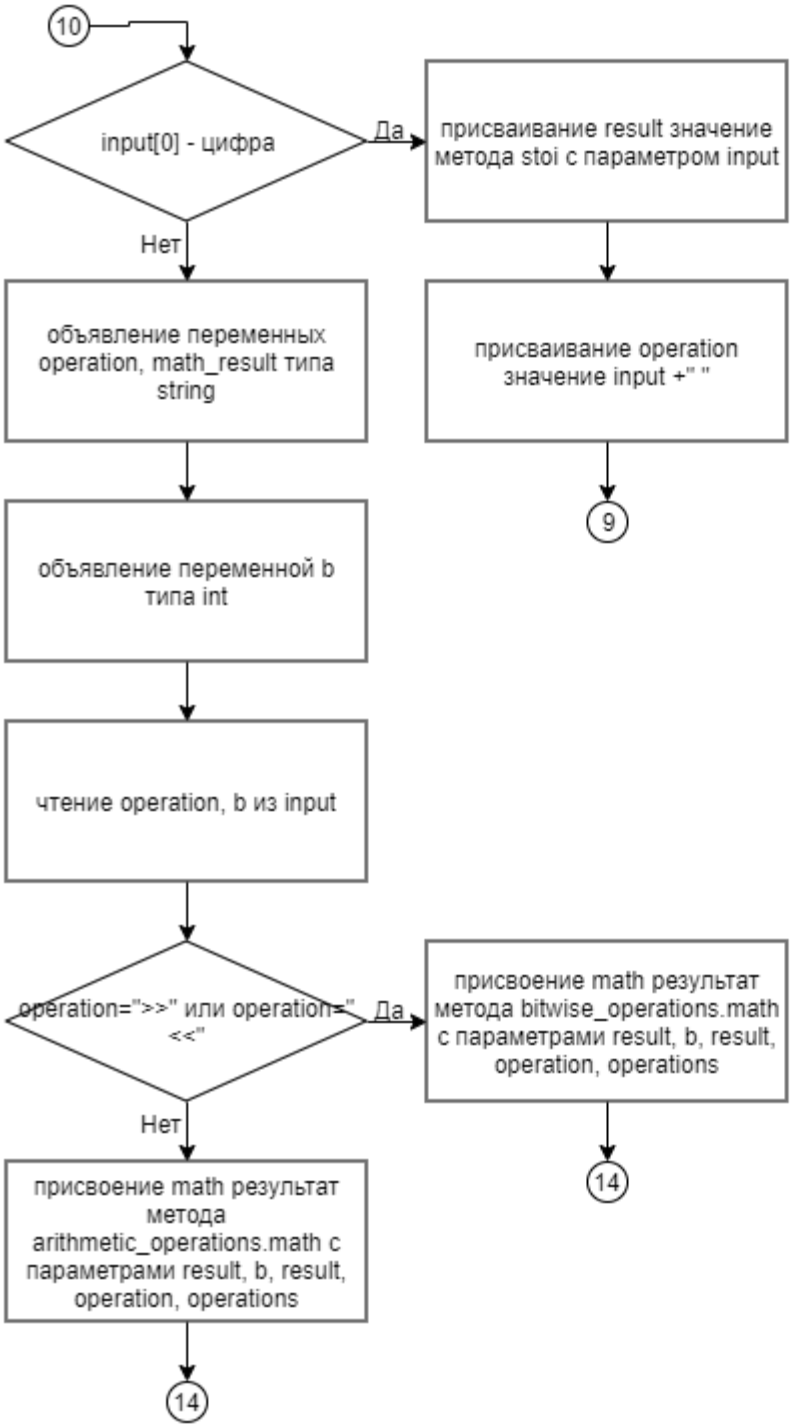

Рисунок 23 – Блок-схема алгоритма

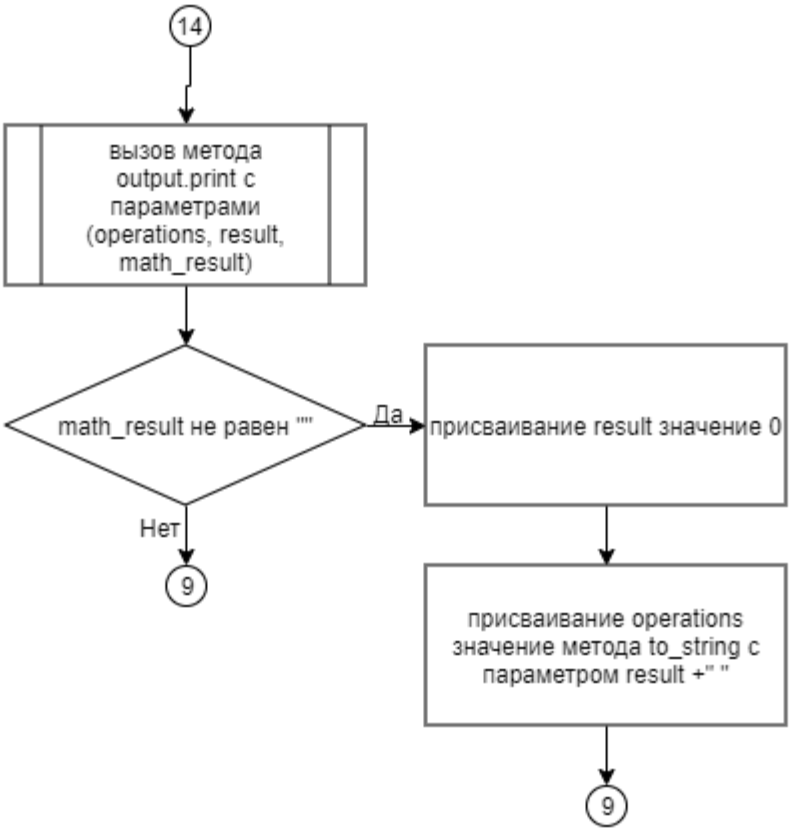

Рисунок 24 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл ArithmeticOperations.cpp

Листинг 1 – ArithmeticOperations.cpp

```
#include "ArithmeticOperations.h"
#include <sstream>

using namespace std;
ArithmeticOperations::ArithmeticOperations(Base*                head_object):
Base(head_object, "ArithmeticOperations") {}
string  ArithmeticOperations::math(short  a,  int  b,  short*  result,  string
operation, string &operations)
{
   operations += operation + " " + to_string(b) + " ";
   if (b > 32767 || b < -32768)
   {
      emit_signal("Overflow");
      return "Overflow";
   }
   if (operation == "+")
   {
      if (__builtin_add_overflow(a, b, result))
      {
         emit_signal("Overflow");
         return "Overflow";
      }
   }
   else if (operation == "-")
   {
      if (__builtin_sub_overflow(a, b, result))
      {
         emit_signal("Overflow");
         return "Overflow";
      }
   }
   else if (operation == "*")
   {
      if (__builtin_mul_overflow(a, b, result))
      {
         emit_signal("Overflow");
         return "Overflow";
      }
   }
```

```
        else if (operation == "/")
        {
           if (b == 0)
           {
              emit_signal("Division by zero");
              return "Division by zero";
           }
           else
           {
              *result = a / b;
           }
        }
        else if (operation == "%")
        {
           if (b == 0)
           {
              emit_signal("Division by zero");
              return "Division by zero";
           }
           else
           {
              *result = a % b;
           }
        }
        emit_signal(to_string(*result));
        return "";
     }
```

5.2 Файл ArithmeticOperations.h

Листинг 2 – ArithmeticOperations.h

```
     #ifndef __ARITHMETICOPERATIONS__H
     #define __ARITHMETICOPERATIONS__H
     #include "Base.h"

     using namespace std;

     class ArithmeticOperations : public Base
     {
        public:
           ArithmeticOperations(Base* head_object);
           string math(short a, int b, short* result, string operation, string
     &operations);
     };

     #endif
```

5.3 Файл Base.cpp

Листинг 3 – Base.cpp

```
#include "Base.h"
#include <iostream>

using namespace std;

bool Base::is_ready()
{
   return state != 0 && (head_object == nullptr || head_object->is_ready());
}
void print_level_padding(int level)
{
   for (int i = 0; i != level; i++)
   cout << " ";
}
Base::Base(Base      *head_object,      string      name)      :      state(0),
name(name),head_object(head_object)
{
   if (head_object)
   {
      head_object->subordinate_objects.push_back(this);
   }
}
Base::~Base()
{
   for (Base *object : subordinate_objects)
   {
      delete object;
   }
}
void Base::print(int level)
{
   print_level_padding(level);
   cout << name << (is_ready() ? " is ready" : " is not ready") << endl;
   for (Base *object : subordinate_objects)
   {

      object->print(level + 1);
   }
}
void Base::set_children_state(int state)
{
   this->state = state;
   for (int i = 0; i != subordinate_objects.size(); i++)
   {
      subordinate_objects[i]->set_children_state(state);
   }
}
void Base::add_signal_handler(Base* object, HANDLER handler)
{
   for (int i = 0; i != handlers.size(); i++)
```

```
        {
            if (handlers[i].object == object && handlers[i].handler == handler)
            {
                return;
            }
        }
        handlers.push_back({object, handler});
    }
    void Base::remove_signal_handler(Base* object, HANDLER handler)
    {
        for (auto it = handlers.begin(); it != handlers.end(); it++)
        {
            if (it->object == object && it->handler == handler)
            {
                handlers.erase(it);
                return;
            }
        }
    }
    void Base::emit_signal(string message)
    {
        if (!is_ready()) return;
        for (int i = 0; i != handlers.size(); i++)
        {
            if (handlers[i].object->is_ready())
            {
                (handlers[i].object->*handlers[i].handler)(message);
            }
        }
    }
```

5.4 Файл Base.h

Листинг 4 – Base.h

```
    #ifndef __BASE__H
    #define __BASE__H
    #include <string>
    #include <vector>

    using namespace std;

    class Base;
    #define HANDLER_D(handler_f)(HANDLER)(&handler_f)
    typedef void (Base::*HANDLER) (std::string);
    class Base
    {
        public:
            Base(Base *head_object, std::string name);
            ~Base();
```

```
        void print(int level = 0);
        bool is_ready();
        void set_children_state(int state);
        void add_signal_handler(Base* object, HANDLER handler);
        void remove_signal_handler(Base* object, HANDLER handler);
        void emit_signal(std::string message);
    private:
        struct connection
        {
            Base* object;
            HANDLER handler;
        };
        int state;
        vector<Base *> subordinate_objects;
        string name;
        Base *head_object;
        vector<connection> handlers;
};

#endif
```

5.5 Файл BitwiseOperations.cpp

Листинг 5 – BitwiseOperations.cpp

```
#include "BitwiseOperations.h"
#include <iostream>

using namespace std;

BitwiseOperations::BitwiseOperations(Base* head_object) : Base(head_object,
"BitwiseOperations") {}
string BitwiseOperations::math(short a, int b, short* result, string
operation, string &operations)
{
    operations += operation + " " + to_string(b) + " ";
    if (b > 32767 || b < -32768)
    {
        emit_signal("Overflow");
        return "Overflow";
    }
    if (operation == "<<")
    {
        *result = a << b;
    }
    else if (operation == ">>")
    {
        *result = a >> b;
    }
    emit_signal(to_string(*result));
```

```
   return "";
}
```

5.6 Файл BitwiseOperations.h

Листинг 6 – BitwiseOperations.h

```
#ifndef __BITWISEOPERATIONS__H
#define __BITWISEOPERATIONS__H
#include "Base.h"

using namespace std;

class BitwiseOperations:public Base
{
   public:
      BitwiseOperations(Base* head_object);
      string math(short a, int b, short* result, string operation, string
&operations);
};

#endif
```

5.7 Файл Input.cpp

Листинг 7 – Input.cpp

```
#include "Input.h"
#include <iostream>

Input::Input(Base* head_object) : Base(head_object, "Input") {}
string Input::read_line()
{
   string input;
   getline(cin, input);
   if (input.size() != 0 && input[input.size() - 1] == '\n')
      input.pop_back();
   if (input.size() != 0 && input[input.size() - 1] == '\r')
      input.pop_back();
   emit_signal(input);
   return input;
}
```

5.8 Файл Input.h

Листинг 8 – Input.h

```
#ifndef __INPUT__H
#define __INPUT__H
#include "Base.h"

using namespace std;

class Input : public Base
{
   public:
      Input(Base* head_object);
      string read_line();
};

#endif
```

5.9 Файл main.cpp

Листинг 9 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "System.h"
int main()
{
   System app;
   app.build_tree_objects();
   return app.exec_app();
}
```

5.10 Файл Output.cpp

Листинг 10 – Output.cpp

```
#include <iostream>
#include <iomanip>
#include <bitset>
#include "Output.h"

using namespace std;
```

```
Output::Output(Base* head_object) : Base(head_object, "Output") {}
void   Output::print(std::string   operations,   short   result,   std::string
last_operation_result)
{
   operations.pop_back();
   cout << operations << "     ";
   if (last_operation_result != "")
   {
      cout << last_operation_result << endl;
      return;
   }
   cout << "HEX " << uppercase << setfill('0') << setw(4) << hex << result
<<dec << "  DEC " << result << "  BIN";
   string bin = bitset<16>(result).to_string();
   for (int i = 0; i != 16; i++)
   {
      if (i % 4 == 0) cout << " ";
         cout << bin[i];
   }
   cout << endl;
}
```

5.11 Файл Output.h

Листинг 11 – Output.h

```
#ifndef __OUTPUT__H
#define __OUTPUT__H
#include "Base.h"

using namespace std;

class Output : public Base
{
   public:
      Output(Base* head_object);
      void     print(string     operations,     short     result,     string
last_operation_result);
};


#endif
```

5.12 Файл Reset.cpp

Листинг 12 – Reset.cpp

```
#include "Reset.h"

using namespace std;

Reset::Reset(Base* head_object) : Base(head_object, "Reset") {}
```

5.13 Файл Reset.h

Листинг 13 – Reset.h

```
#ifndef __RESET__H
#define __RESET__H
#include "Base.h"

using namespace std;

class Reset : public Base
{
   public:
      Reset(Base* head_object);
};


#endif
```

5.14 Файл System.cpp

Листинг 14 – System.cpp

```
#include <iostream>
#include <sstream>
#include "System.h"
#include "Input.h"
#include "ArithmeticOperations.h"

using namespace std;

System::System() : Base(nullptr, "System") {}
void System::build_tree_objects()
```

```
{
   input = new Input(this);
   arithmetic_operations = new ArithmeticOperations(this);
   output = new Output(this);
   bitwise_operations = new BitwiseOperations(this);
}
int System::exec_app()
{
   set_children_state(1);
   string input, operations;
   short result;
   while (input = this->input->read_line(), input != "Off")
   {
      if (input == "SHOWTREE")
      {
         print();
      }
      else if (input == "C")
      {
         emit_signal("C");
      }
      else if (isdigit(input[0]))
      {
         result = stoi(input);
         operations = input + " ";
      }
      else
      {
         stringstream input_stream(input);
         string operation, math_result;
         int b;
         input_stream >> operation >> b;
         if (operation == ">>" || operation == "<<")
         {
            math_result   =   bitwise_operations->math(result,   b,   &result,
operation, operations);
         }
         else
         {
            math_result  =  arithmetic_operations->math(result,  b,  &result,
operation, operations);
         }
         output->print(operations, result, math_result);
         if (math_result != "")
         {
         result = 0;
         operations = to_string(result) + " ";
         }
      }
   }
   return 0;
}
```

5.15 Файл System.h

Листинг 15 – System.h

```
#ifndef __SYSTEM__H
#define __SYSTEM__H
#include "Base.h"
#include "Input.h"
#include "ArithmeticOperations.h"
#include "Output.h"
#include "BitwiseOperations.h"

using namespace std;

class System : public Base
{
   public:
      System();
      void build_tree_objects();
      int exec_app();
   private:
      Input* input;
      ArithmeticOperations* arithmetic_operations;
      Output* output;
      BitwiseOperations* bitwise_operations;
};


#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 23.

Таблица 23 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
5 + 5 << 1 / 0 + 5 C 7 8 / -3 C 9 % -4 + 7 * 11 Off	5 + 5 HEX 000A DEC 10 BIN 0000 0000 0000 1010 5 + 5 << 1 HEX 0014 DEC 20 BIN 0000 0000 0001 0100 5 + 5 << 1 / 0 Division by zero 0 + 5 HEX 0005 DEC 5 BIN 0000 0000 0000 0101 8 / -3 HEX FFFE DEC -2 BIN 1111 1111 1111 1110 9 % -4 HEX 0001 DEC 1 BIN 0000 0000 0000 0001 9 % -4 + 7 HEX 0008 DEC 8 BIN 0000 0000 0000 1000 9 % -4 + 7 * 11 HEX 0058 DEC 88 BIN 0000 0000 0101 1000	5 + 5 HEX 000A DEC 10 BIN 0000 0000 0000 1010 5 + 5 << 1 HEX 0014 DEC 20 BIN 0000 0000 0001 0100 5 + 5 << 1 / 0 Division by zero 0 + 5 HEX 0005 DEC 5 BIN 0000 0000 0000 0101 8 / -3 HEX FFFE DEC -2 BIN 1111 1111 1111 1110 9 % -4 HEX 0001 DEC 1 BIN 0000 0000 0000 0001 9 % -4 + 7 HEX 0008 DEC 8 BIN 0000 0000 0000 1000 9 % -4 + 7 * 11 HEX 0058 DEC 88 BIN 0000 0000 0101 1000
5 + 3 C 3 * 9 C SHOWTREE Off	5 + 3 HEX 0008 DEC 8 BIN 0000 0000 0000 1000 3 * 9 HEX 001B DEC 27 BIN 0000 0000 0001 1011 System is ready Input is ready ArithmeticOperations is ready Output is ready BitwiseOperations is ready	5 + 3 HEX 0008 DEC 8 BIN 0000 0000 0000 1000 3 * 9 HEX 001B DEC 27 BIN 0000 0000 0001 1011 System is ready Input is ready ArithmeticOperations is ready Output is ready BitwiseOperations is ready

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы по моделированию работы инженерного калькулятора была реализована в полном объеме. Полученный результат работы показывает работу инженерного калькулятора. Работа инженерного калькулятора включает в себя вычисления целых чисел, объемом 2 байта.

Парадигма объектно-ориентированного программирования полезна и удобна в сфере программирования своим функционалом. В объектно-ориентированных языках программирования легче реализована модульность, организация программы на совокупность небольших блоков. Модернизация программы является ключом к успешной программы. ООП объединяет данные и связанное с ними поведение в объект, что помогает программистам легче понять, как работает код.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).