Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель курсовой работы - это моделирование работы логического калькулятора, используя сигналы и обработчики

Чтобы выполнить поставленную цель необходимо выполнить следующие цели:

Освоение объектно-ориентированногго програмирования

Освоение объектно-ориентированного языка программирования C++

Закрепление навыка разработки программы

Освоение выполнения всех необходимых работ согласно этапам разработки программы и соответствующих программных инструментов

Освоение умения разработки программы как системы

Освоение умения проектирования архитектуры программы на базе построения иерархии объектов

Освоение навыка программирования по заранее определенным правилам

Освоение весионности при разраотке программ

Использование единой базы данных

Тестирование работы программы

1 ПОСТАНОВКА ЗАДАЧИ

Авторы задачи магистр группы ИВМО-01-20 Люлява Даниил и студент группы ИВБО-01-18 Дуксин Никита.

Разработать программу, которой на вход подается последовательность пар строк:

- строка, содержащая логическую функцию в инфиксной форме. Операнды и операции разделены пробелом. Признаком конца формулы служит точка, перед которой пробел не ставится.
- строка со значениями логических переменных в этой формуле. В качестве значений логических переменных подается либо «0», либо «1».

В функции могут быть использованы следующие операции: AND – конъюнкция, OR – дизъюнкция, XOR – исключающее «ИЛИ», NOT – инверсия, => – импликация, <=> – эквивалентность. Приоритет операций согласно правилам математической логики.

Считается, что ошибок в инфиксной форме не будет, равно как и все переменные получат корректные значения.

Признаком завершения ввода будет являться строка, состоящая из точки.

Необходимо в самом начале вывести на экран строку «OUT», а затем с новой строки значения переменных.

Затем преобразовать полученную строку формулы в обратную польскую нотацию (в качестве разделителя операндов и операций – 1 пробел). Вывести полученный результат на экран.

Затем на основании значений пропозициональных (логических) переменных, введенных с клавиатуры, вычислить полученное выражение, основываясь на сформированной обратной польской нотации и вывести результат вычислений на экран.

Помимо команд обусловленных постановкой задачи, необходимо предусмотреть возможность вывода на экран созданного дерева объектов с отметкой о их готовности. Команда, которая должна за это отвечать - "SHOWTREE". После Вывода дерева на экран программа должна завершиться.

Использовать объекты:

1. Для ввода очередной строки и считывания значений переменных

Объект выдает следующие сигналы:

- инициирующий считывание строки с формулой
- инициирующий формирование множества пропозициональных (логических) переменных
- инициирующий вывод значений переменных
- инициирующий формирование польской нотации

2. Для формирования множества логических переменных из строки

Объект выдает следующие сигналы:

- инициирующий ввод значений логических переменных

3. Для формирования обратной польской нотации логической функции

Объект выдает сигнал инициирующий вывод обратной польской нотации и подсчета значения функции

4. Для вывода сообщений на экран.

Функционал:

- Вывод значений логических переменных. Возможные значения переменных при выводе: «true», «false»
- Вывод сформированной обратной польской нотации
- Вывод результата вычисления функции

5. Для подсчета значения функции

- Объект выдает сигнал, инициирующий вывод результата вычислений

Все взаимодействия между объектами организовать посредством сигналов и

обработчиков.

Алгоритм формирования обратной польской нотации должен использовать стек. Алгоритм вычисления результата также должен использовать стек.

Написать программу, реализующую следующий алгоритм:

1. Вывод на экран строки «OUT». Переход к пункту 2.

2. Выдача сигнала на считывание строки логической функции. Переход к пункту 3.

3. Если введенная строка состоит из точки, то выход, иначе переход в пункт 4.

4. Выдача сигнала на формирование множества логических переменных. Переход к пункту 5.

5. Выдача сигнала на вывод значений переменных. Переход к пункту 6.

6. Выдача сигнала на формирование польской нотации. Переход к пункту 2.

1.1 Описание входных данных

Каждая нечетная строка, начиная с первой:

«логическая функция в инфиксной форме»

или

«.»

Каждая четная строка, согласно шаблону:

«имя логической переменной»␣=␣«значение переменной»␣«имя логической переменной»␣=␣«значение переменной»…

Пример:

c <=> NOT (a XOR b OR a AND c) => b XOR c.

a = 0 b = 1 c = 1

.

1.2 Описание выходных данных

OUT

Values:␣«имя логической переменной»␣=␣«значение переменной»␣«имя логической переменной»␣=␣«значение переменной»…

Polish Notation:␣«сформированная обратная польская нотация логической функции»

Result:␣«результат вычисления функции»

…

Пример:

OUT

Values: a = false b = true c = true

Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=>

Result: true

2 МЕТОД РЕШЕНИЯ

Для решения поставленной задачи были изменены следующие классы, методы:

Класс cl_2 переименован в Reader

Класс cl_3 переименован в Printer

Класс cl_4 переименован в TransformToPN

Класс cl_5 переименован в Values

Класс cl_6 переименован в Compute

Удален метод signal_1

Удален метод handler_2

Удален метод signal_3

Указатель на сигнал был изменен с указателя метода, чье входные параметры были ссылкой на строковую переменную на указатель метода, чье входные данные равны ссылкой на строковую переменную и ссылкой на словарь(ключ - строковая переменная, значение - логическая переменная)

Указатель на обработчик был изменен с указателя метода, чье входные параметры были на строковую переменную на указатель метода, чье входные данные равны строковая переменная и словарь(ключ - строковая переменная, значение - логическая переменная)

Метод signal_2 переименован в signal_pn_2, также входные данные были изменены с ссылки на строку на ссылку строки и ссылку словаря(ключ - строковая переменная, значение - логическая переменная)

Метод signal_4 переименован в signal_send_4, также входные данные были изменены с ссылки на строку на ссылку строки и ссылку словаря(ключ - строковая переменная, значение - логическая переменная)

Входные данные метода handler_4 были изменены со строковой переменной, на

строковую переменную и словарь(ключ - строковая переменная, значение - логическая переменная)

Метод signal_5 переименован в signal_send_5, также входные данные были изменены с ссылки на строку на ссылку строки и ссылку словаря(ключ - строковая переменная, значение - логическая переменная)

Входные данные метода handler_5 были изменены со строковой переменной, на строковую переменную и словарь(ключ - строковая переменная, значение - логическая переменная)

Входные данные метода signal_6 были изменены с ссылки на строковую переменную, на ссылку на строковую переменную и ссылку на словарь(ключ - строковая переменная, значение - логическая переменная)

Входные данные метода handler_6 были изменены со строковой переменной, на строковую переменную и словарь(ключ - строковая переменная, значение - логическая переменная)

Входные данные метода emit_signal были изменены с указателя метода сигнала и строковую переменную на указатель метода сигнала

Входные данные метода get_signal были изменены с указателя на BaseClass на указатель на BaseClass и логическую переменную

Для решения поставленной задачи были добавлены следующие библиотеки:

1. Библиотека sstream, подключающая структуру данных istringstream
2. Библиотека stack, подключающая структуру данных stack и методы push, empty, top, pop к этой структуру
3. Библиотека map, подключающая структуру данных map

Класс Application:

- Методы:

5.0handler_1

1. Функционал: Проверка на завершения работы программы

5.1build_tree_objects

Функционал: Создает дерево объектов, а также связи между объектами

5.2exec_app

Функционал: Вызывов сигналов и обработчиков у объектов

Класс Reader:

Методы:

signal_pn_2

Функционал: Ввод данных

signal_values_2

Функционал: Ввод данных

Класс Printer:

Методы:

handler_3

Функционал: Вывод данных

Класс TransformToPN:

Поля

Поле, отвечающее за запись логического выражения в польскую запись

Наименование - pn

Тип - строковый

Модификатор доступа - закрытый

Методы:

signal_send_4

Фунционал: Создание текста о польской нотации для передачи в другие объекты

signal_print_4

Функционал: Создание текста о польской нотации для вывода

handler_4

Функционал: Изменение записи логического выражения на польскую запись

Класс Values:

Поля:

Поле, отвечающее за хранение множества логических переменных из логического выражения

Наименование - arguments

Тип - словарь(ключ - строковая переменная, значение - логическая переменная)

Модификатор доступа - закрытый

Методы:

signal_send_5

Фунционал: Создание списка переменных удобного для считывания другими объектами

signal_print_5

Функционал: Создание текста о данных переменных для удобного вывода

handler_5

Функционал: Создание множества логических переменных, принимающих участие в логическом выражении

Класс Compute:

Поля:

Поле, отвечающее за хранение значения логического выражения

Наименование - ans

Тип - логический

Модификатор доступа - закрытый

Поле, отвечающее за запись логического выражения в польскую запись

Наименование - pn

Тип - строковый

Модификатор доступа - закрытый

Поле, отвечающее за хранение множества логических переменных из логического

выражения

Наименование - arguments

Тип - словарь(ключ - строковая переменная, значение - логическая переменная)

Модификатор доступа - закрытый

Методы:

singnal_6

Фунционал: Создание текста о результате логического выражения удобного для вывода

handler_6

Фунционал: Вычисление результата логического выражения

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 метода emit_signal класса BaseClass

Функционал: Вызывает обработчики классов, связных с этим классом.

Параметры: Указатель на метод сигнала: p_signal.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода emit_signal класса BaseClass

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной s_command: ""	2
2		Инициализация пустого словаря(ключ - строковая переменная, значение - логическая переменная) values	3
3		Объявление указателя на метод обработчик p_handler	4
4		Инициализация на указатель класса BaseClass p_object: пустой	5
5		Вызов метода сигнала у текущего объекта: s_command, values	6
6		Инициализация целочисленной переменной i: 0	7
7	i меньше чем размер connects		8
			∅
8	Указатель на сигнал	Присвоение p_handler: указатель на обрабочик у	9

№	Предикат	Действия	№ перехода
	connects[i] совпадает с p_signal	connects[i]	
			11
9		Присвоение p_object: p_base у connects[i]	10
10		Вызов метода обрабочика у p_object: s_command, values	11
11		Присвоение значения i: i + 1	7

3.2 Алгоритм метода get_signal класса BaseClass

Функционал: Поиск указателя на метод сигнала.

Параметры: Указатель на BaseClass: object, логический: choice .

Возвращаемое значение: Указатель на метод сигнала.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода get_signal класса BaseClass

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной object_class: вызов функции name у функции typeid(указатель на object)	2
2	В object_class есть подстрока "Reader"		3
			4
3	choice равен 1	Вернуть указатель класса Reader на метод сигнала signal_pn_2	∅
		Вернуть указатель класса Reader на метод сигнала signal_values_2	∅
4	В object_class есть подстрока		5

№	Предикат	Действия	№ перехода
	"TransfromToPN"		
			6
5	choice равен 1	Вернуть указатель класса TransformToPN на метод сигнала signal_send_4	∅
		Вернуть указатель класса TransformToPN на метод сигнала signal_print_4	∅
6	В object_class есть подстрока "Values"		7
		Вернуть указатель класса Compute на метод сигнала signal_6	∅
7	choice равен 1	Вернуть указатель класса Values на метод сигнала signal_send_5	∅
		Вернуть указатель класса Values на метод сигнала signal_print_5	∅

3.3 Алгоритм метода get_handler класса BaseClass

Функционал: Поиск указателя на метод обработчика.

Параметры: Указатель на BaseClass: object.

Возвращаемое значение: Указатель на метод обработчика.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода get_handler класса BaseClass

№	Предикат	Действия	№ перехода
1		Инициализация строковой переменной object_class: вызов функции name у функции typeid(указатель на object)	2
2	В object_class есть подстрока "Application"	Вернуть указатель класса Application на метод обработчика handler_1	∅

№	Предикат	Действия	№ перехода
			3
3	В object_class есть подстрока "Printer"	Вернуть указатель класса Printer на метод обработчика handler_3	∅
			4
4	В object_class есть подстрока "TransfromToPN"	Вернуть указатель класса TransformToPN на метод обработчика handler_4	∅
			5
5	В object_class есть подстрока "Values"	Вернуть указатель класса Values на метод обработчика handler_5	∅
		Вернуть указатель класса Compute на метод обработчика handler_6	∅

3.4 Алгоритм метода build_tree_objects класса Application

Функционал: Создает дерево объектов, а также связи между объектами.

Параметры: Нет.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода build_tree_objects класса Application

№	Предикат	Действия	№ перехода
1		Инициализация указателя на BaseClass read: Reader(текущий объект, "Reader")	2
2		Инициализация указателя на BaseClass print: Reader(текущий объект, "Printer")	3
3		Инициализация указателя на BaseClass values: Reader(текущий объект, "Values")	4
4		Инициализация указателя на BaseClass pn: Reader(текущий объект, "TransformToPN")	5

№	Предикат	Действия	№ перехода
5		Инициализация указателя на BaseClass calc: Reader(текущий объект, "Compute")	6
6		Вызов метода set_connect(вызов метода get_signal(read, 1) от текущего объекта, текущий объект, вызов метода get_handler(текущий объект) от текущего объекта) у read	7
7		Вызов метода set_connect(вызов метода get_signal(read, 1) от текущего объекта, pn, вызов метода get_handler(pn) от текущего объекта) у read	8
8		Вызов метода set_connect(вызов метода get_signal(read, 0) от текущего объекта, values, вызов метода get_handler(values) от текущего объекта) у read	9
9		Вызов метода set_connect(вызов метода get_signal(values, 1) от текущего объекта, calc, вызов метода get_handler(calc) от текущего объекта) у values	10
10		Вызов метода set_connect(вызов метода get_signal(values, 0) от текущего объекта, print, вызов метода get_handler(print) от текущего объекта) у values	11
11		Вызов метода set_connect(вызов метода get_signal(pn, 1) от текущего объекта, calc, вызов метода get_handler(calc) от текущего объекта) у pn	12
12		Вызов метода set_connect(вызов метода get_signal(pn, 0) от текущего объекта, print, вызов метода get_handler(print) от текущего объекта) у pn	13
13		Вызов метода set_connect(вызов метода get_signal(calc) от текущего объекта, print, вызов метода get_handler(print) от текущего объекта) у calc	∅

3.5 Алгоритм метода exec_app класса Application

Функционал: Вызывов сигналов и обработчиков у объектов.

Параметры: Нет.

Возвращаемое значение: Целочисленное: 0. Индикатор корректности завершения программы.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода exec_app класса Application

№	Предикат	Действия	№ перехода
1		Инициализация указателя на BaseClass read: вызов get_object_by_name("Reader") у текущего объекта	2
2		Инициализация указателя на BaseClass values: вызов get_object_by_name("Values") у текущего объекта	3
3		Инициализация указателя на BaseClass pn: вызов get_object_by_name("TransformToPN") у текущего объекта	4
4		Инициализация указателя на BaseClass calc: вызов get_object_by_name("Compute") у текущего объекта	5
5		Вывод на экран: "OUT"	6
6	Правда		7
		Вернуть 0	∅
7		Вызов метода emit_signal(вызов метода get_signal(read, 1) у текущего объекта) у read	8
8		Вызов метода emit_signal(вызов метода get_signal(read, 1) у текущего объекта) у pn	9
9		Вызов метода emit_signal(вызов метода get_signal(read, 0) у текущего объекта) у read	10
10		Вызов метода emit_signal(вызов метода get_signal(values, 1) у текущего объекта) у values	11
11		Вызов метода emit_signal(вызов метода get_signal(values, 0) у текущего объекта) у values	12

№	Предикат	Действия	№ перехода
12		Вызов метода emit_signal(вызов метода get_signal(pn, 0) у текущего объекта) у pn	13
13		Вызов метода emit_signal(вызов метода get_signal(calc) у текущего объекта) у calc	6

3.6 Алгоритм метода signal_pn_2 класса Reader

Функционал: Ввод данных.

Параметры: Ссылка на строковую переменную: text, ссылка на словарь(ключ - строковый, значение - логическое).

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода signal_pn_2 класса Reader

№	Предикат	Действия	№ перехода
1		Ввод с помощью клавиатуры: text	∅

3.7 Алгоритм метода signal_values_2 класса Reader

Функционал: Ввод данных.

Параметры: Ссылка на строковую переменную: text, ссылка на словарь(ключ - строковый, значение - логическое).

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода signal_values_2 класса Reader

№	Предикат	Действия	№ перехода
1		Ввод с помощью клавиатуры: text	∅

3.8 Алгоритм метода handler_3 класса Printer

Функционал: Вывод данных.

Параметры: Строковая переменная: text, Словарь(ключ - строковый, значение - логическое).

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода handler_3 класса Printer

№	Предикат	Действия	№ перехода
1		Вывод на экран: переход на следующию строку, text	∅

3.9 Алгоритм метода signal_print_4 класса TransformToPN

Функционал: Создание текста о польской нотации для вывода.

Параметры: Ссылка на строковую переменную: text, ссылка на словарь(ключ - строковый, значение - логическое).

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода signal_print_4 класса TransformToPN

№	Предикат	Действия	№ перехода
1		Присвоение text: "Polish Notation" + pn	∅

3.10 Алгоритм метода signal_send_4 класса TransformToPN

Функционал: Создание текста о польской нотации для передачи в другие объекты.

Параметры: Ссылка на строковую переменную: text, ссылка на словарь(ключ - строковый, значение - логическое).

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода signal_send_4 класса TransformToPN

№	Предикат	Действия	№ перехода
1		Присвоение text: pn	∅

3.11 Алгоритм метода handler_4 класса TransformToPN

Функционал: Изменение записи логического выражения на польскую запись.

Параметры: Строковая переменная: text, Словарь(ключ - строковый, значение - логическое).

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода handler_4 класса TransformToPN

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной word	2
2		Объявление стека, содержащего строковые переменные, current_vals	3
3		Инициализация потока formula: от начала text до предпоследнего элемента text	4
4		Инцилизация словаря(ключ - строковый, значение - логическое) actions: {{"(", 1}, {")", 1}, {"<=>", 2}, {"=>", 3}, {"XOR", 4}, {"OR", 4}, {"AND", 5}, {"NOT", 6} }	5
5		Присвоение pn: ""	6
6	word не пустой		7

№	Предикат	Действия	№ перехода
			20
7	Длина word равна 0	Пропустить итерацию цикла	6
			8
8		Инициализация логической переменной is_exist: 0	9
9		Инициализация итератора el: начало actions	10
10	el не равен концу actions		11
			13
11	el.first равен word	Присвоение is_exist: 1	12
			12
12		Присвоение el: Сместить el на 1 ближе к концу actions	10
13	is_exist равен 0	Присвоение pn: pn + " " + word	6
			14
14	word равен "("	Вызов метода push(word) у current_vals	6
			15
15	word равен ")"		16
			18
16	current_vals не пустой и текущий элемент у current_vals не равен "("	Присвоение pn: pn + " " + текущий элемент у current_vals	17
		Удалить текущий элемент у current_vals	18
17		Удалить текущий элемент у current_vals	6
18	current_vals не пустой и actions[текущиего элемента current_vals] больше либо равен actions[word]	Присвоение pn: pn + " " + текущий элемент у current_vals	19
		Добавить элемент word в current_vals	6
1		Удалить текущий элемент у current_vals	18

№	Предикат	Действия	№ перехода
9			
20	current_vals не пустой	Присвоение pn: pn + " " + текущий элемент у current_vals	21
			∅
21		Удалить текущий элемент у current_vals	20

3.12 Алгоритм метода signal_send_5 класса Values

Функционал: Создание списка переменных удобного для считывания другими объектами.

Параметры: Ссылка на строковую переменную: text, ссылка на словарь(ключ - строковый, значение - логическое) values.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода signal_send_5 класса Values

№	Предикат	Действия	№ перехода
1		Присвоение values: arguments	∅

3.13 Алгоритм метода signal_print_5 класса Values

Функционал: Создание текста о данных переменных для удобного вывода.

Параметры: Ссылка на строковую переменную: text, ссылка на словарь(ключ - строковый, значение - логическое) values.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода signal_print_5 класса Values

№	Предикат	Действия	№ перехода
1		Присвоение values: "Values:"	2
2		Инициализация итератора el: начало arguments	3
3	el не равен концу arguments		4
			∅
4	el.second равен 1	Присвоение values: values + " " + el.first + " = true"	5
		Присвоение values: values + " " + el.first + " = false"	5
5		Сдвиг el на 1 ближе к концу arguments	3

3.14 Алгоритм метода signal_6 класса Compute

Функционал: Создание текста о результате логического выражения удобного для вывода.

Параметры: Ссылка на строковую переменную: text, ссылка на словарь(ключ - строковый, значение - логическое) values.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода signal_6 класса Compute

№	Предикат	Действия	№ перехода
1		Присвоение text: "Result: "	2
2	ans равен 1	Присвоение text: text + "true"	∅
		Присвоение text: text + "false"	∅

3.15 Алгоритм метода handler_6 класса Compute

Функционал: Вычисление результата логического выражения.

Параметры: Строковая переменная: text, Словарь(ключ - строковый,

значение - логическое) values.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода handler_6 класса Compute

№	Предикат	Действия	№ перехода
1	Длина text больше 0	Присвоение pn: text	2
			2
2	values не пустой	Присвоение arguments: values	3
			3
3	pn не пустой и arguments не пустой	Объявление стека(информационная часть - логическая переменная) current_vals	4
			∅
4		Объявление потока ss: pn + " "	5
5		Объявление строковой переменной word	6
6	word не пуст		7
		Присвоение ans: текущий элемент current_vals	∅
7	word пустой	Пропустить итерацию цилка	6
			8
8		Инцилизация логической переменной is_exist: 0	9
9		Инициализация итератора el: начало arguments	10
10	el не равен концу arguments		11
			13
11	el.first равен word	Присвоение is_exist: 1	12
			12
12		Сдвиг el на 1 ближе к концу	10
13	is_exist равен 1	Добавить word в current_vals	6
			14

№	Предикат	Действия	№ перехода
14		Объявление логических переменных: a, b	15
15	word не равен "NOT"	Присвоение b: текущий элемент current_vals	16
			23
16		Удаление текущего элемента из current_vals	17
17		Присвоение a: текущий элемент current_vals	18
18		Удаление текущего элемента из current_vals	19
19	word равен "AND"	Добавить (a * b) в current_vals	6
			20
20	word равен "OR"	Добавить (a или b) в current_vals	6
			21
21	word равен =>"	Добавить (!a или b) в current_vals	6
			22
22	word равен <=>"	Добавить (a равен b) в current_vals	6
		Добавить (a ^ b) в current_vals	6
23		Присвоение a: текущий элемент current_vals	24
24		Удаление текущего элемента из current_vals	25
25		Добавление (!a) в current_vals	6

3.16 Алгоритм метода handler_1 класса Application

Функционал: Проверка на завершения работы программы.

Параметры: Строковая переменная: text, Словарь(ключ - строковый,

значение - логическое).

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода handler_1 класса Application

№	Предикат	Действия	№ перехода
1	text равен "SHOWTREE"	Вызов метода print_tree_ready(0) у текущего объекта	2
			2
2	text равен "SHOWTREE" или равен "."	Завершение работы программы	∅
			∅

3.17 Алгоритм метода handler_5 класса Values

Функционал: Создание множества логических переменных, принимающих участие в логическом выражении.

Параметры: Строковая переменная: text, словарь(ключ - строковый, значение - логическое) values.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler_5 класса Values

№	Предикат	Действия	№ перехода
1		Присвоение arguments: пустота	2
2		Инициализация потока iss: text	3
3		Объявление строковых переменных: word, name	4
4		Объявление логической переменной value	5
5		Инициализация целочисленной переменной tmp: 0	6
6	word не пуст	Присвоение tmp: tmp + 1	7

№	Предикат	Действия	№ перехода
			∅
7	tmp равен 1	Присвоение name: word	6
			8
8	tmp равен 3		9
			6
9	word равен "0"	Присвоение value: 0	10
		Присвоение value: 1	10
10		Присвоение arguments[name]: value	11
11		Присвоение tmp: 0	6

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-13.

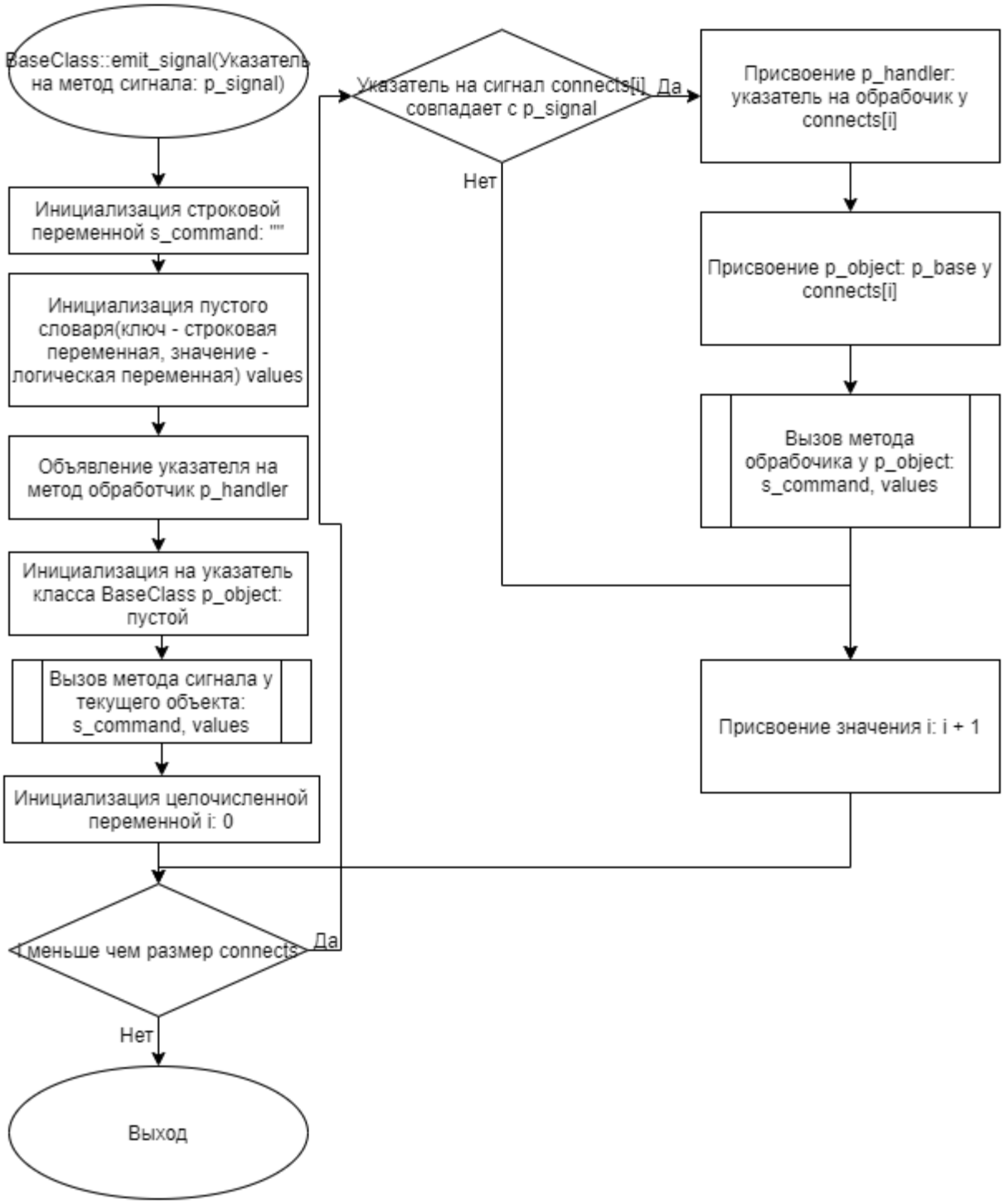

Рисунок 1 – Блок-схема алгоритма

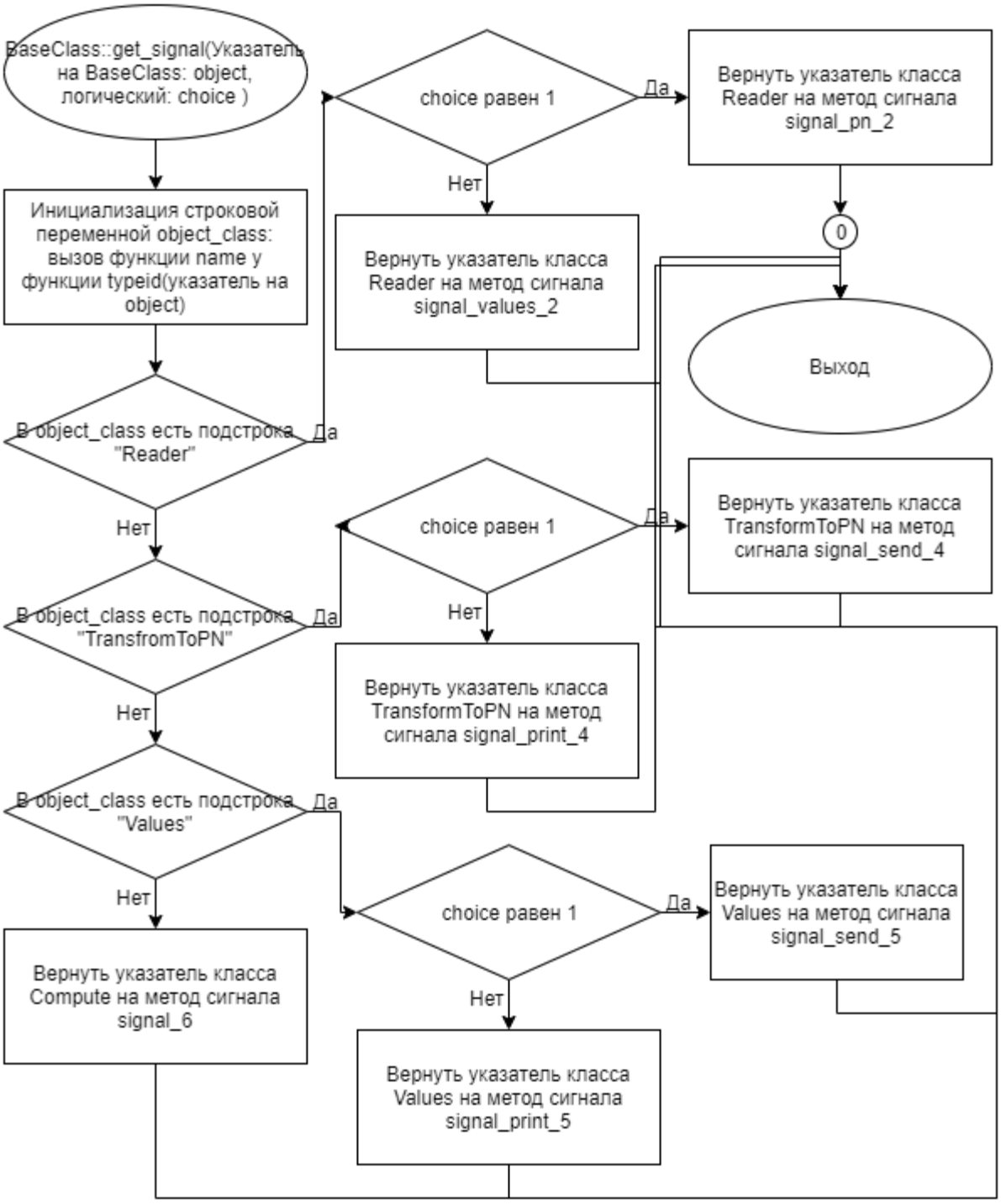

Рисунок 2 – Блок-схема алгоритма

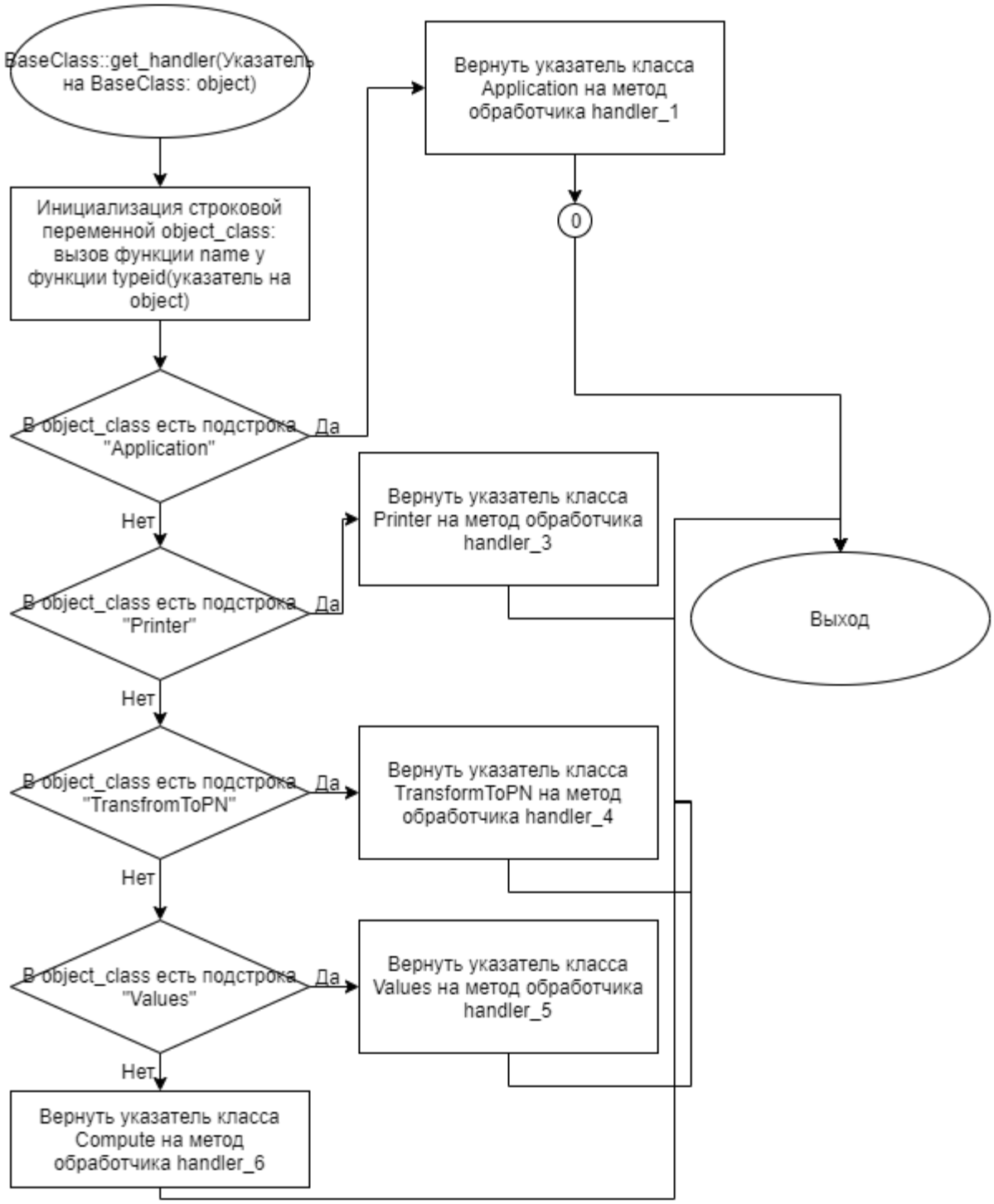

Рисунок 3 – Блок-схема алгоритма

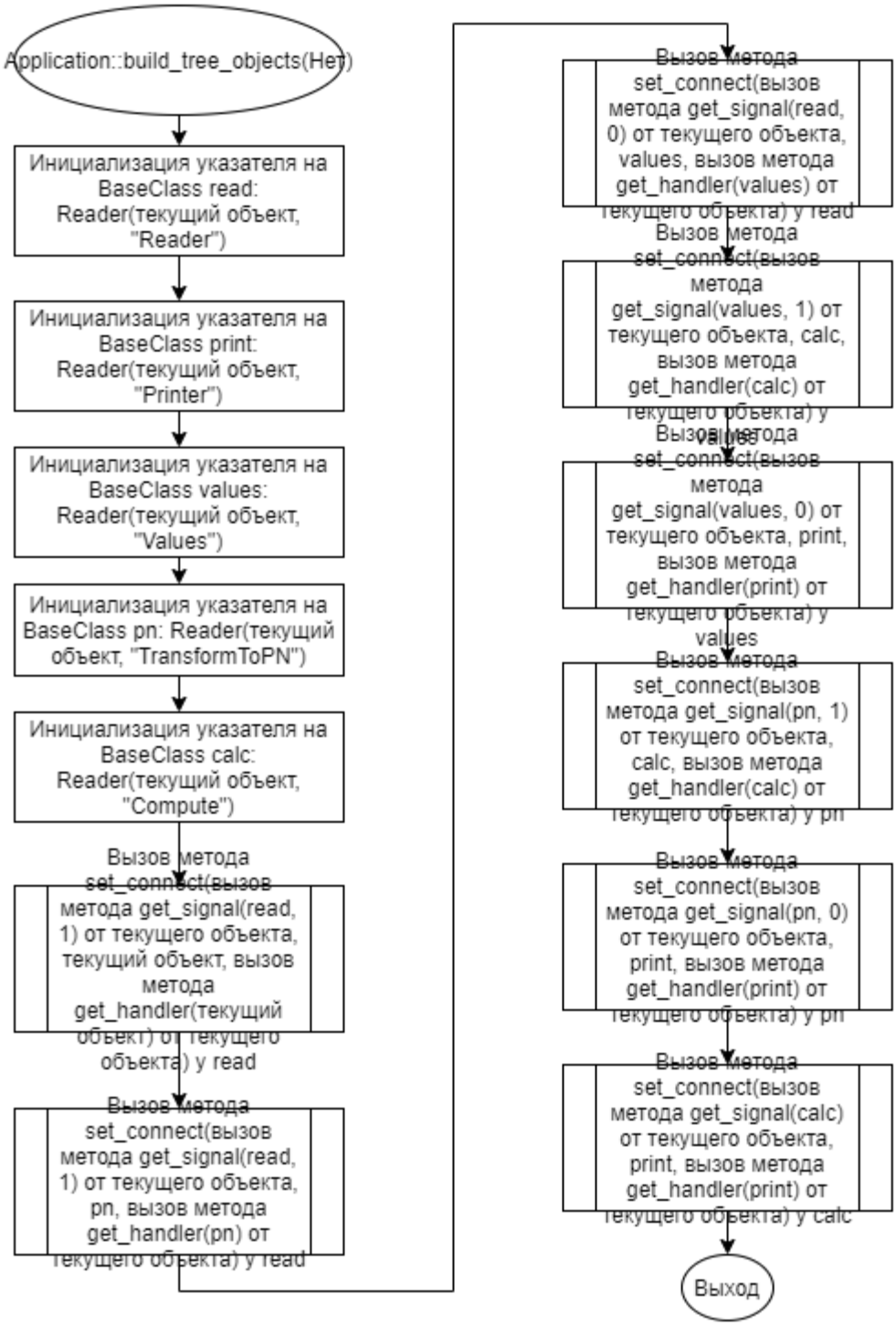

Рисунок 4 – Блок-схема алгоритма

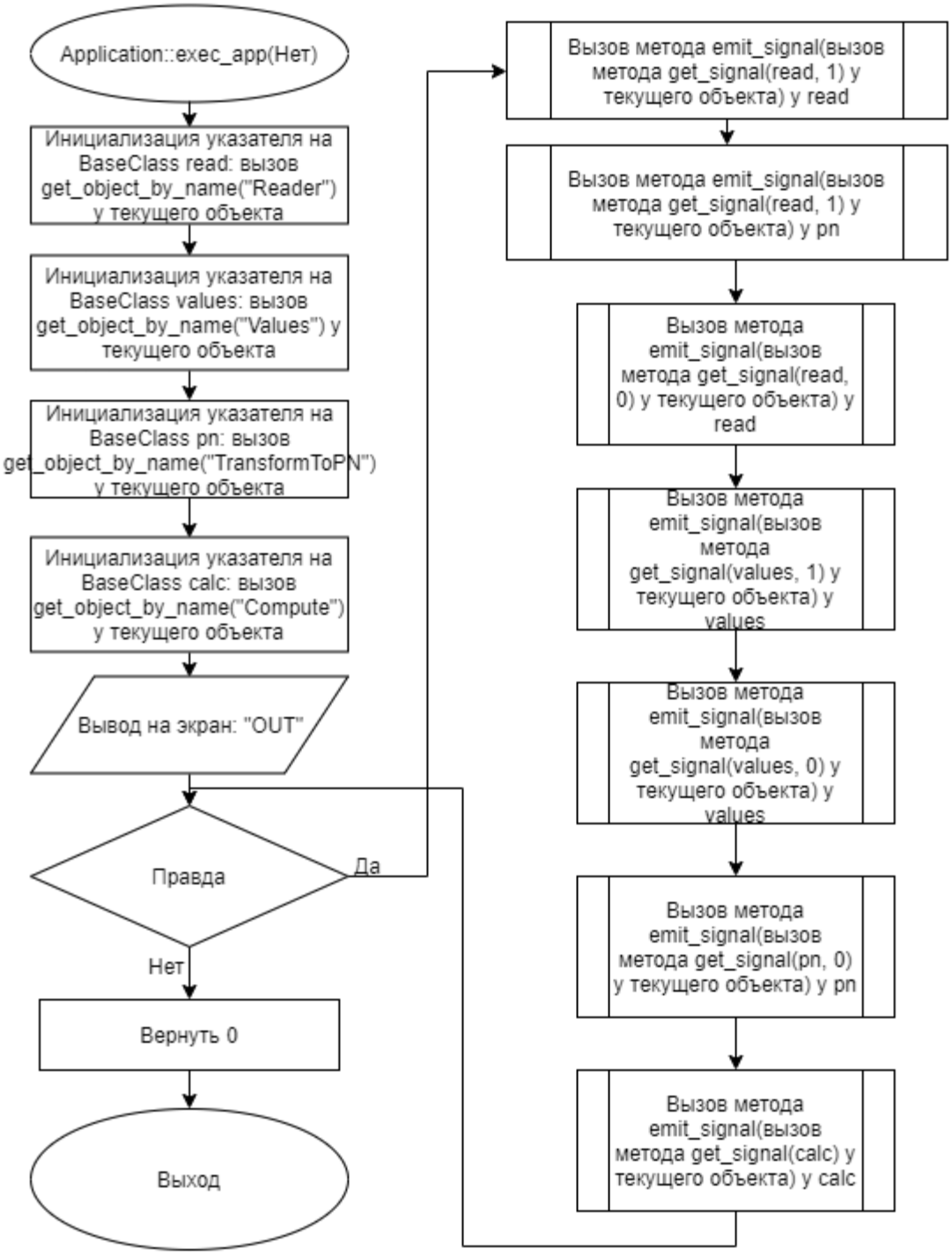

Рисунок 5 – Блок-схема алгоритма

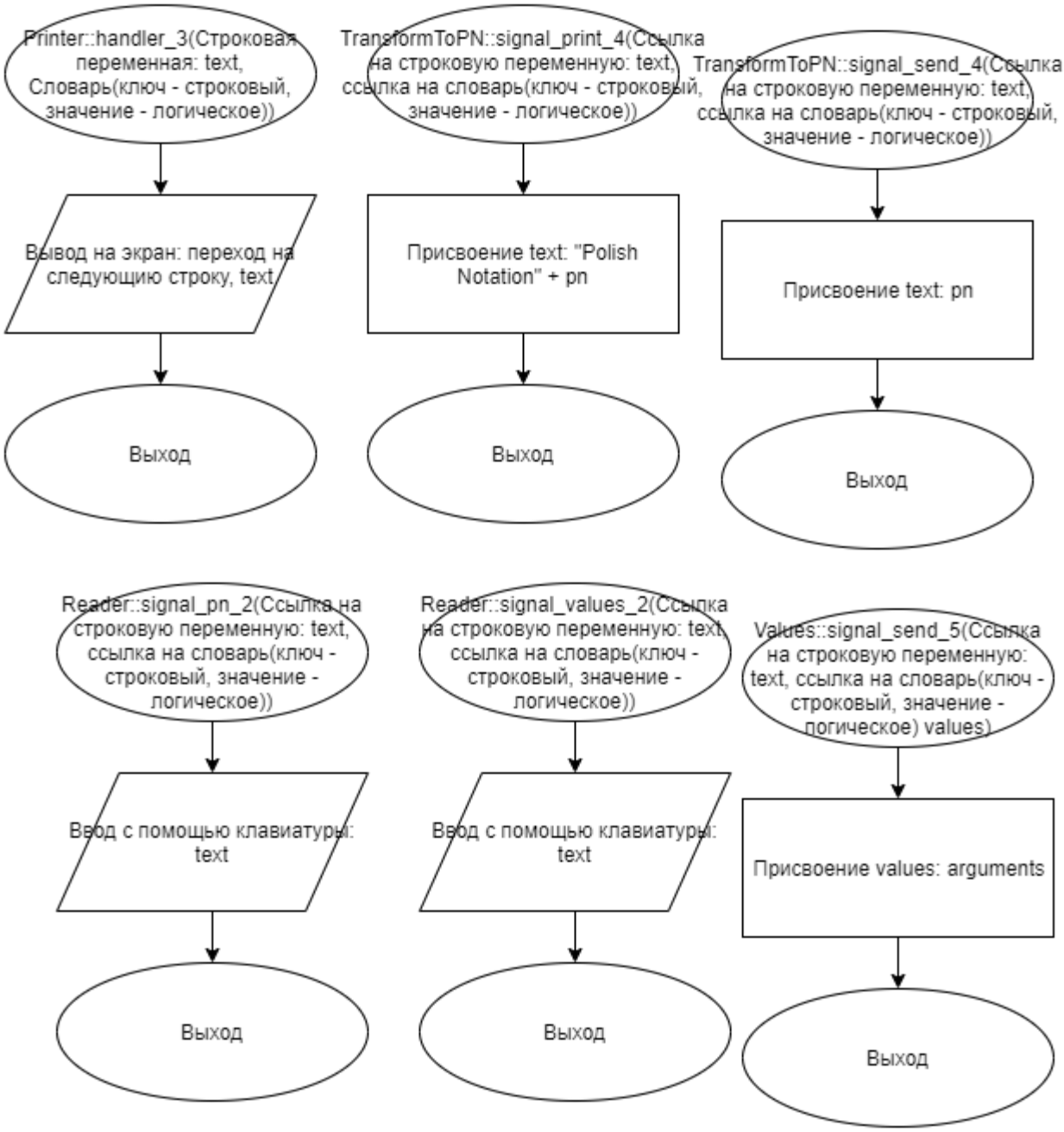

Рисунок 6 – Блок-схема алгоритма

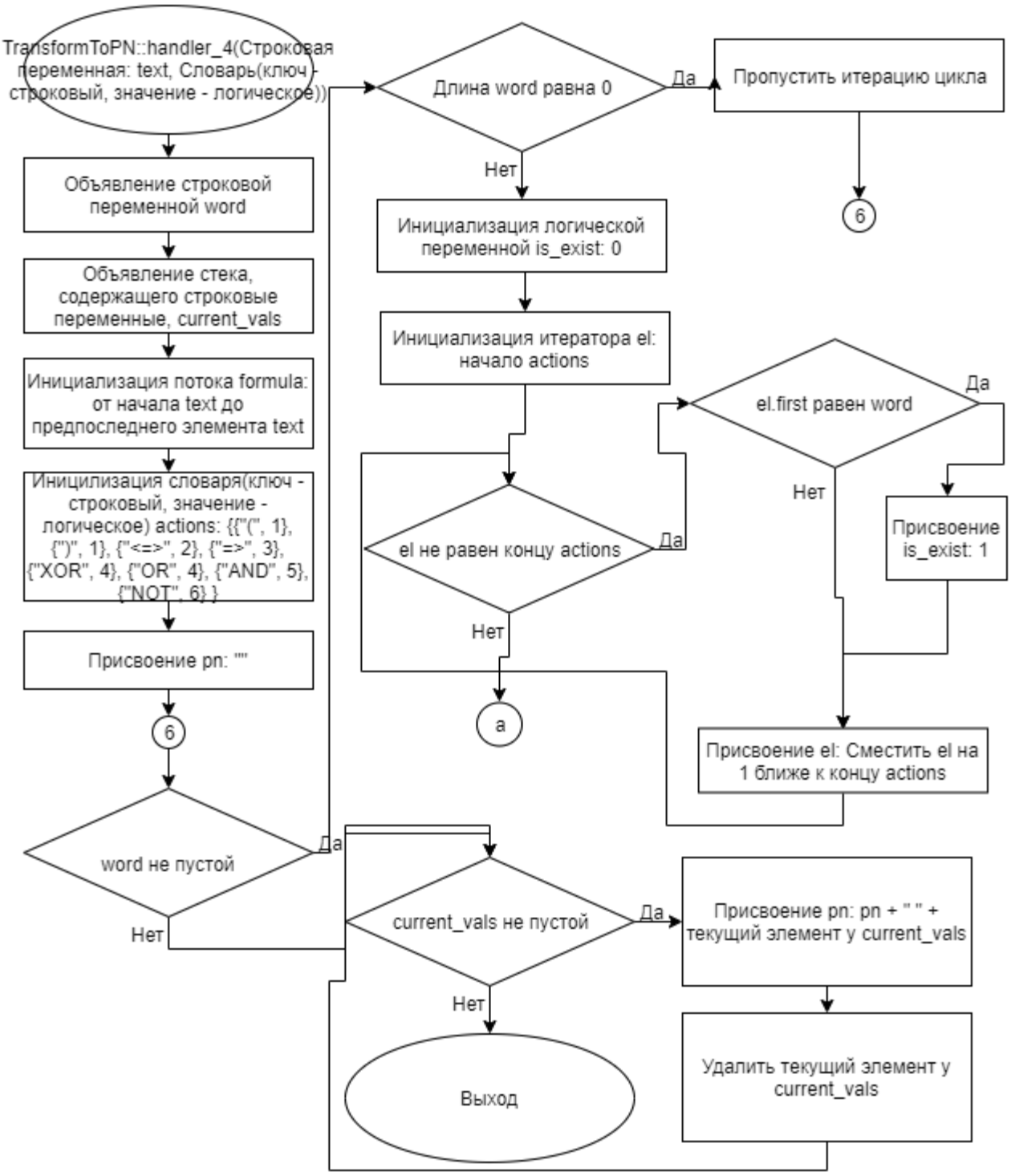

Рисунок 7 – Блок-схема алгоритма

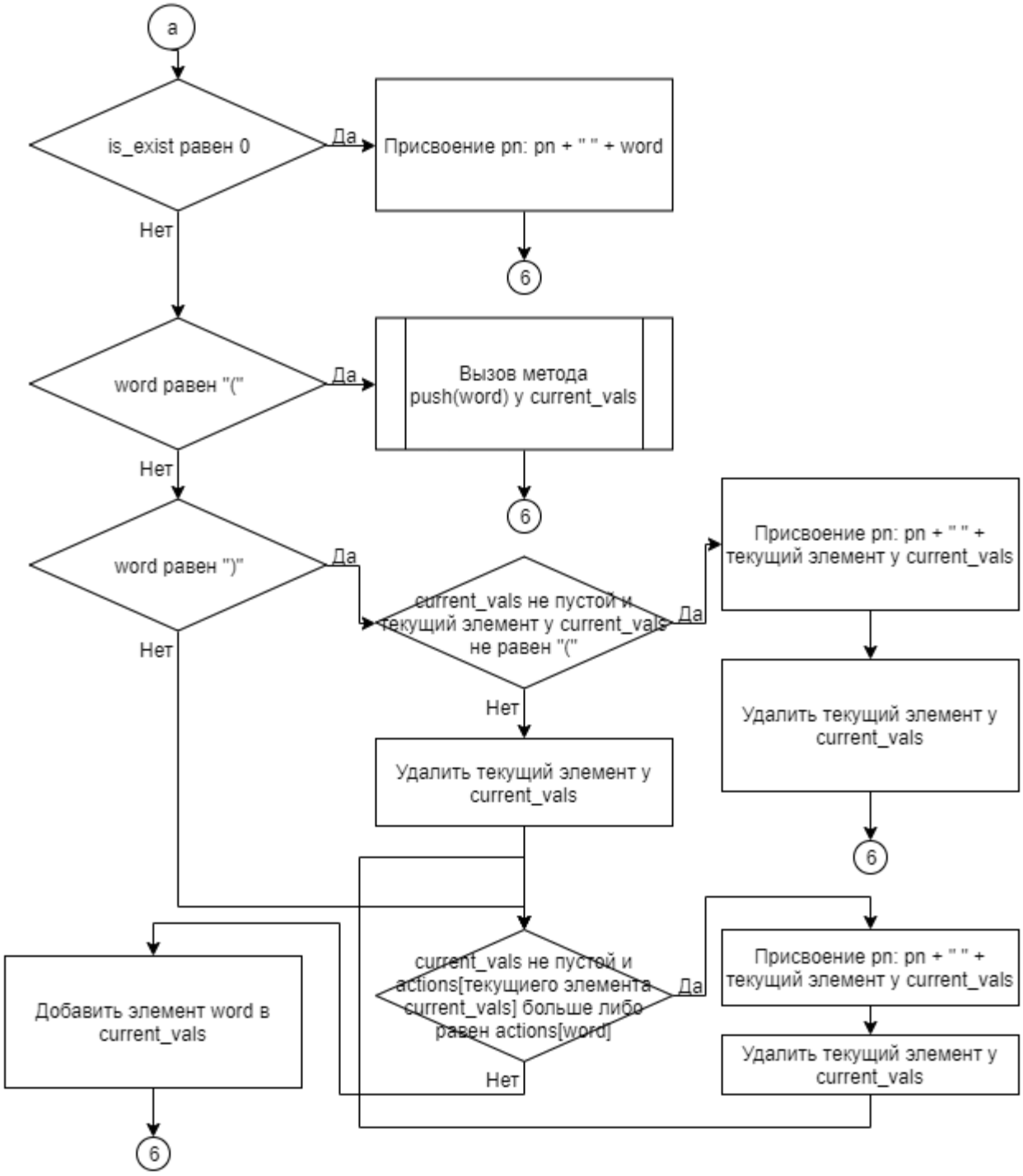

Рисунок 8 – Блок-схема алгоритма

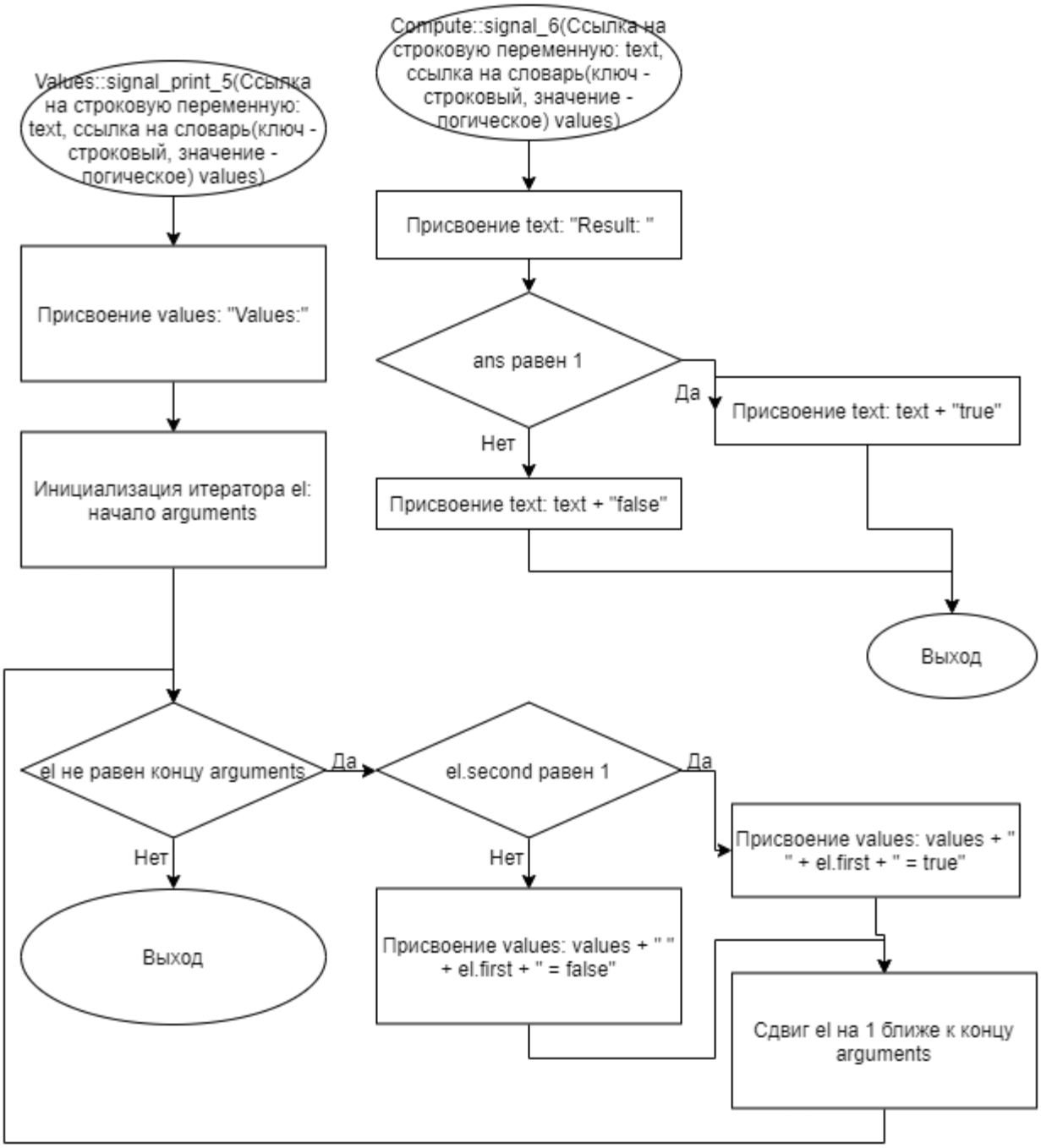

Рисунок 9 – Блок-схема алгоритма

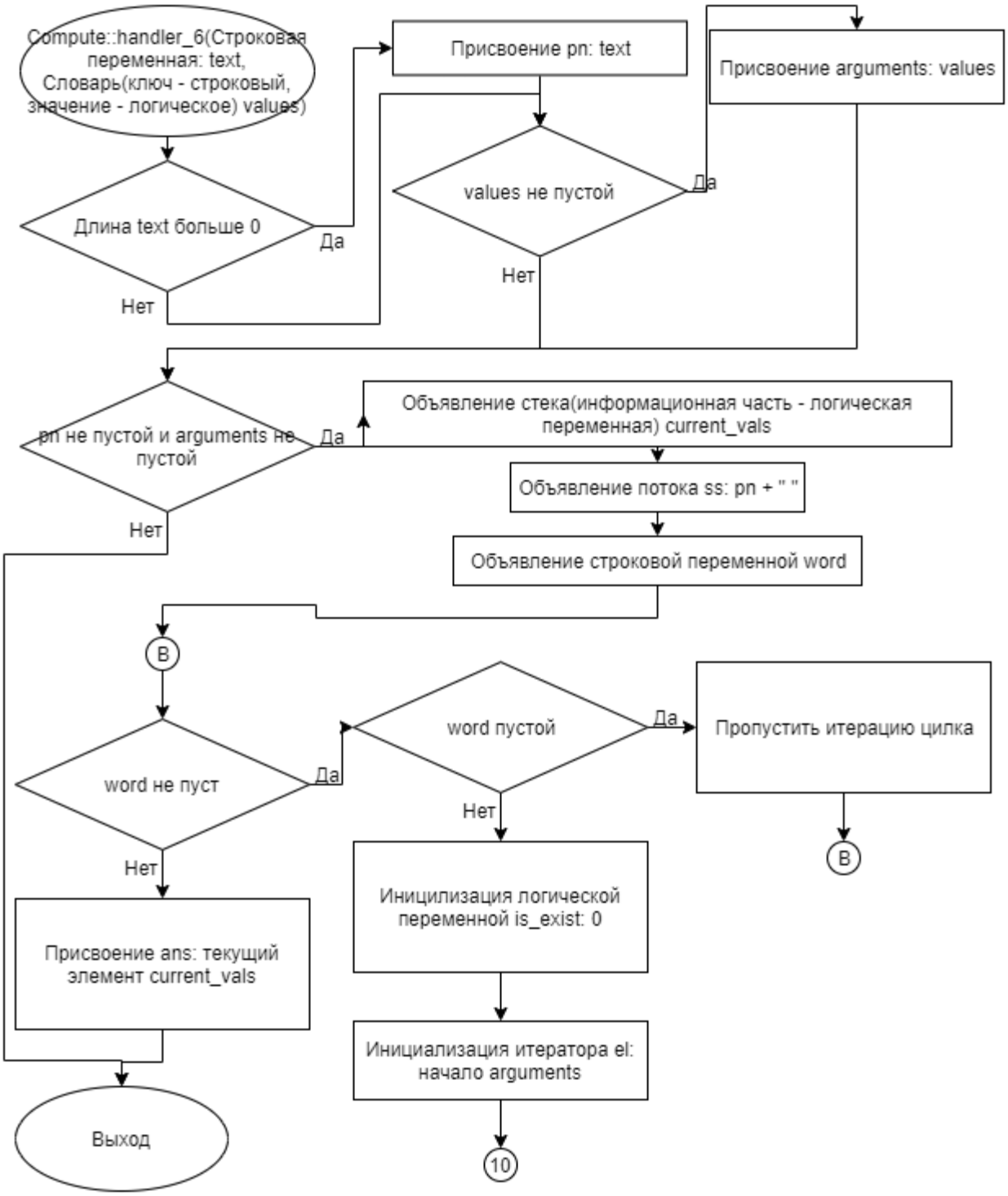

Рисунок 10 – Блок-схема алгоритма

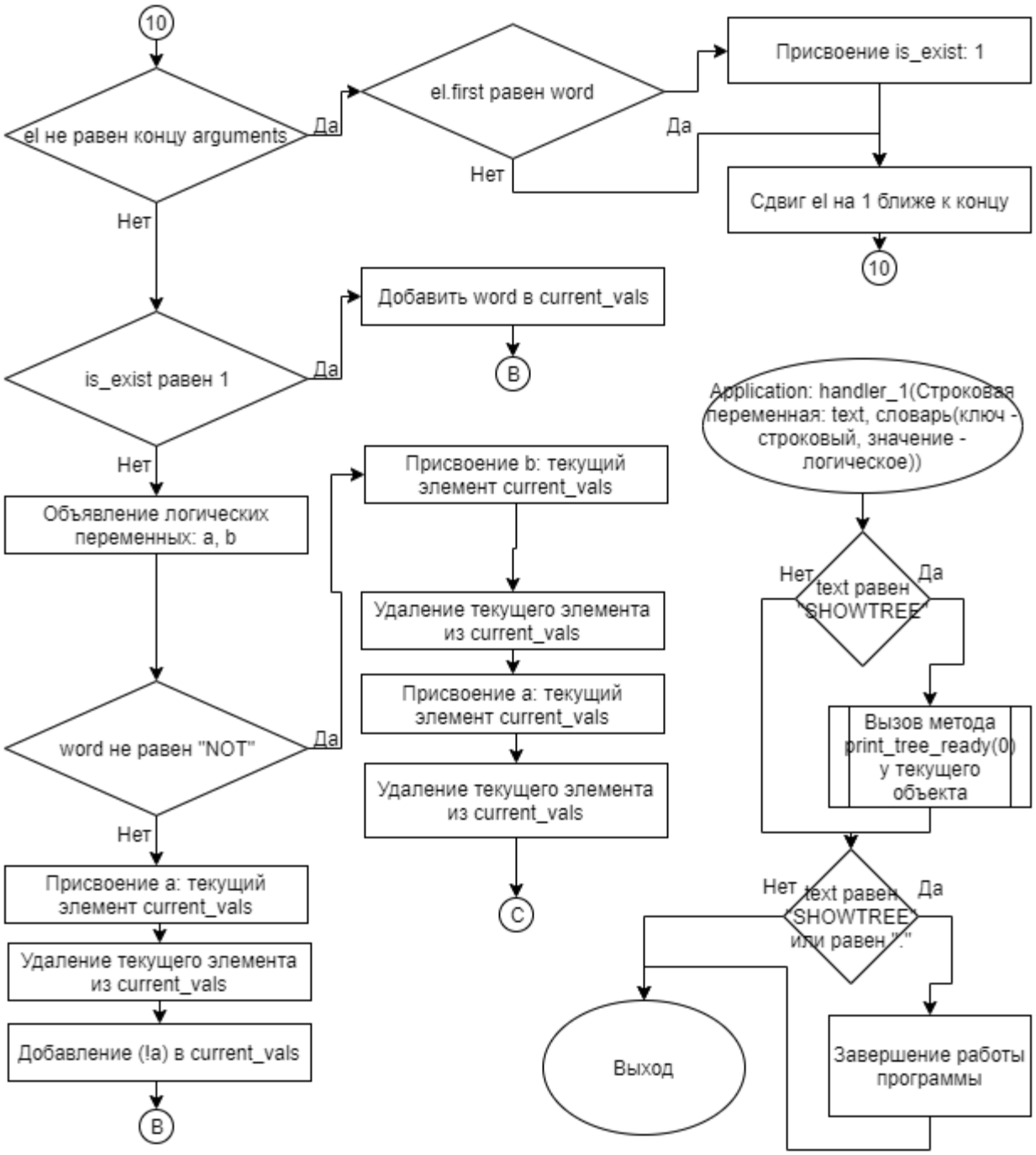

Рисунок 11 – Блок-схема алгоритма

Рисунок 12 – Блок-схема алгоритма

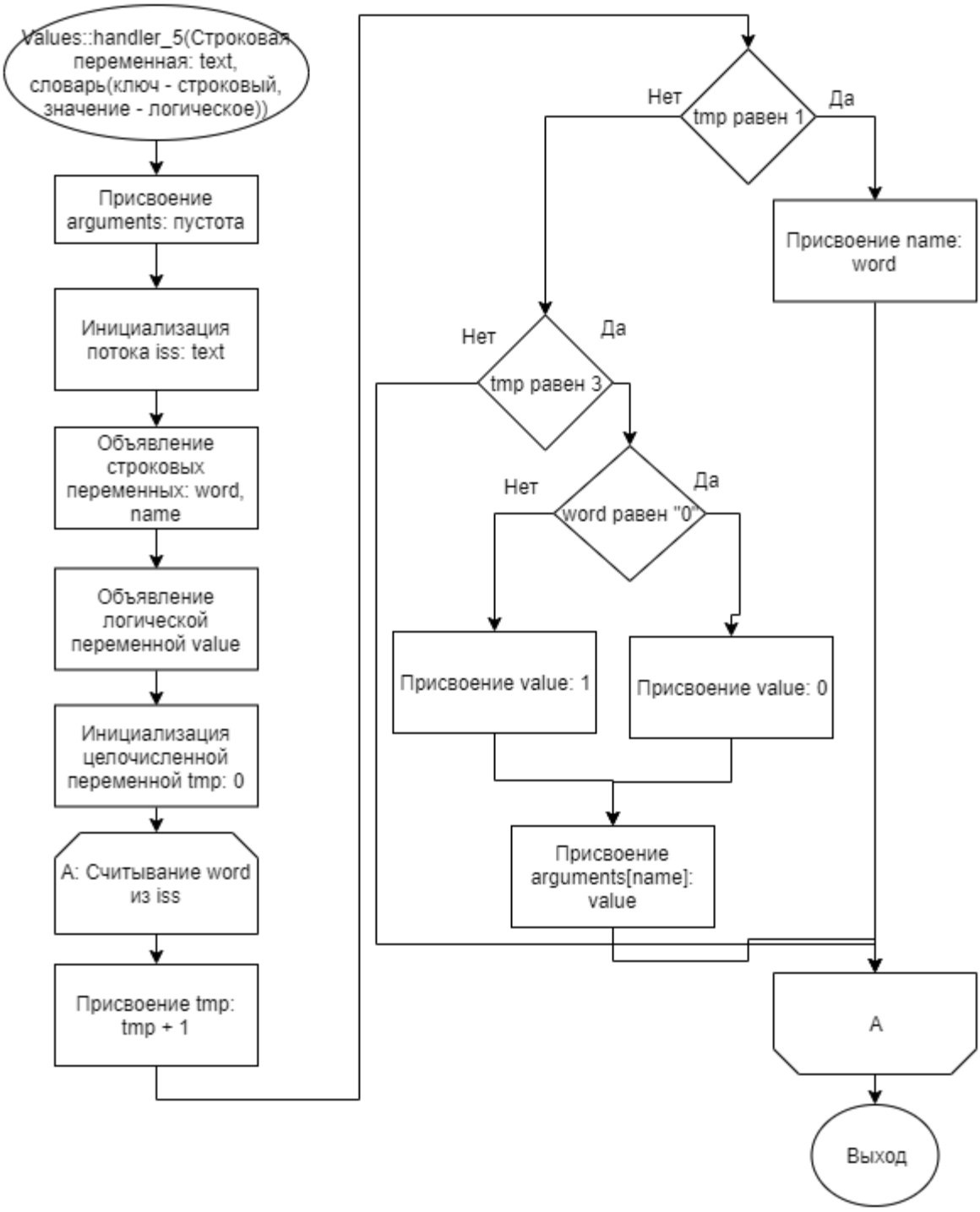

Рисунок 13 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.3 Файл Application.cpp

Листинг 1 – Application.cpp

```
#include "Application.h"
#include "Reader.h"
#include "Printer.h"
#include "TransformToPN.h"
#include "Values.h"
#include "Compute.h"

void Application::build_tree_objects(){
	BaseClass* read = new Reader(this, "Reader");
	BaseClass* print = new Printer(this, "Printer");
	BaseClass* values = new Values(this, "Values");
	BaseClass* pn = new TransformToPN(this, "TransformToPN");
	BaseClass* calc = new Compute(this, "Compute");
	read->set_connect(get_signal(read, 1), this, get_handler(this));
	read->set_connect(get_signal(read, 1), pn, get_handler(pn));
	read->set_connect(get_signal(read, 0), values, get_handler(values));
	values->set_connect(get_signal(values, 1), calc, get_handler(calc));
	values->set_connect(get_signal(values, 0), print, get_handler(print));
	pn->set_connect(get_signal(pn, 1), calc, get_handler(calc));
	pn->set_connect(get_signal(pn, 0), print, get_handler(print));
	calc->set_connect(get_signal(calc), print, get_handler(print));
}

int Application::exec_app(){
	BaseClass* read = this->get_object_by_name("Reader");
	BaseClass* values = this->get_object_by_name("Values");
	BaseClass* pn = this->get_object_by_name("TransformToPN");
	BaseClass* calc = this->get_object_by_name("Compute");
	cout << "OUT";
	while (true){
		read->emit_signal(get_signal(read, 1));
		pn->emit_signal(get_signal(pn, 1));
		read->emit_signal(get_signal(read, 0));
		values->emit_signal(get_signal(values, 1));
		values->emit_signal(get_signal(values, 0));
		pn->emit_signal(get_signal(pn, 0));
		calc->emit_signal(get_signal(calc));
	}
	return 0;
}
```

```
void Application::handler_1(string text, map<string, bool>){
	if (text == "SHOWTREE")
		this->print_tree_ready(0);
	if (text == "." || text == "SHOWTREE")
		exit(0);
}
```

5.4 Файл Application.h

Листинг 2 – Application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H
#include "BaseClass.h"

class Application: public BaseClass{
private:
	bool work = 1;
public:
	using BaseClass::BaseClass;
	void build_tree_objects();
	int exec_app();
	void handler_1(string, map<string, bool>);
};

#endif
```

5.5 Файл BaseClass.cpp

Листинг 3 – BaseClass.cpp

```
#include "BaseClass.h"
#include "Application.h"
#include "Reader.h"
#include "Printer.h"
#include "TransformToPN.h"
#include "Values.h"
#include "Compute.h"

BaseClass::BaseClass(BaseClass * head, string object_name){
	set_name(object_name);
	set_header(head);
	if (head != nullptr)
		head->subordinate_objects.push_back(this);
}

string BaseClass::get_name(){
	return name;
}
```

```
void BaseClass::set_ready(int status) {
	if (status != 0 && (header == nullptr || header->ready != 0))
		ready = 1;
	else{
		ready = 0;
		for (auto el: subordinate_objects)
			el->set_ready(0);
	}
}

int BaseClass::get_ready(){
	return ready;
}

void BaseClass::set_name(string new_name){
	name = new_name;
}

void BaseClass::set_header(BaseClass* new_head){
	header = new_head;
}

BaseClass* BaseClass::get_header(){
	return header;
}

void BaseClass::print_tree(int indent){
	for (int i = 0; i < indent; ++i)
		cout << " ";
	cout << name << "\n";
	for (int i = 0; i < subordinate_objects.size(); ++i)
		subordinate_objects[i]->print_tree(indent + 4);
}

void BaseClass::print_tree_ready(int indent){
	cout << "\n";
	for (int i = 0; i < indent; ++i)
		cout << " ";
	cout << name;
		if (ready)
			cout << " is ready";
		else
			cout << " is not ready";
	for (int i = 0; i < subordinate_objects.size(); ++i)
		subordinate_objects[i]->print_tree_ready(indent + 4);
}

BaseClass* BaseClass::get_object_by_name(string find_name){
	if (name == find_name)
		return this;
	for (auto &el: subordinate_objects){
		if (el->get_name() == find_name)
			return el;
		BaseClass* ans = el->get_object_by_name(find_name);
		if (ans)
			return ans;
```

```
	}
	return nullptr;
}

BaseClass* BaseClass::find_way(string find_name, BaseClass* current){
	string object_name = "";
	int i = 0;
	if (find_name == ".")
		return current;
	else if (find_name == "/"){
		return this;
	}
	else if (find_name[0] == '/' && find_name[1] == '/'){
		for (i = 2; i < find_name.size(); ++i)
			object_name += find_name[i];
		return this->get_object_by_name(object_name);
	}
	if (find_name[0] == '/'){
		current = this;
		i = 1;
	}
	else
		i = 0;
	for (; i < find_name.size(); ++i){
		if (find_name[i] == '/' || i == find_name.size() - 1){
			if (i == find_name.size() - 1)
				object_name += find_name[i];
			bool check = 0;
			for (auto el: current->subordinate_objects){
				if (el->get_name() == object_name){
					current = el;
					check = 1;
				}
			}
			object_name = "";
			if (!check)
				return nullptr;
		}
		else
			object_name += find_name[i];
	}
	return current;
}

string BaseClass::get_absolute_way(){
	string way = "";
	if (!this->header)
		way = "/";
	BaseClass* object = this;
	while (object->header){
		way = "/" + object->get_name() + way;
		object = object->get_header();
	}
	return way;
}

BaseClass::~BaseClass(){
```

```
}

void    BaseClass::set_connect(TYPE_SIGNAL    p_signal,    BaseClass*    p_object,
TYPE_HANDLER p_ob_handler){
      for (int i = 0; i < connects.size();++i)
            if  (connects[i]->p_signal  ==  p_signal  &&  connects[i]->p_base  ==
p_object && connects[i]->p_handler == p_ob_handler)
                  return;
      o_sh* p_value = new o_sh;
      p_value->p_signal = p_signal;
      p_value->p_base = p_object;
      p_value->p_handler = p_ob_handler;
      connects.push_back(p_value);
}

void   BaseClass::delete_connect(TYPE_SIGNAL   p_signal,   BaseClass*   p_object,
TYPE_HANDLER p_ob_handler){
      for (int i = 0; i < connects.size(); ++i){
            if  (connects[i]->p_signal  ==  p_signal  &&  connects[i]->p_base  ==
p_object && connects[i]->p_handler == p_ob_handler){
                  connects.erase(connects.begin() + i);
                  return;
            }
      }
}

void BaseClass::emit_signal(TYPE_SIGNAL p_signal){
      string s_command = "";
      map<string, bool> values = {};
      TYPE_HANDLER p_handler;
      BaseClass *p_object;
      (this->*p_signal)(s_command, values);
      for (int i = 0; i < connects.size(); ++i){
            if (connects[i]->p_signal == p_signal && connects[i]->p_base->ready !=
0){
                  p_handler = connects[i]->p_handler;
                  p_object = connects[i]->p_base;
                  (p_object->*p_handler)(s_command, values);
            }
      }
}

TYPE_SIGNAL BaseClass::get_signal(BaseClass* object, bool choice){
      string object_class = typeid(*object).name();
      if (object_class.find("Reader") != string::npos){
            if (choice)
                  return SIGNAL_D(Reader:: signal_pn_2);
            else
                  return SIGNAL_D(Reader:: signal_values_2);
      }
      else if (object_class.find("TransformToPN") != string::npos){
            if (choice)
                  return SIGNAL_D(TransformToPN:: signal_send_4);
            else
                  return SIGNAL_D(TransformToPN:: signal_print_4);
      }
      else if (object_class.find("Values") != string::npos){
```

```
            if (choice)
                  return SIGNAL_D(Values:: signal_send_5);
            else
                  return SIGNAL_D(Values:: signal_print_5);
      }
      else
            return SIGNAL_D(Compute:: signal_6);
}

TYPE_HANDLER BaseClass::get_handler(BaseClass* object){
      string object_class = typeid(*object).name();
      if (object_class.find("Application") != string::npos)
            return HANDLER_D(Application:: handler_1);
      else if (object_class.find("Printer") != string::npos)
            return HANDLER_D(Printer:: handler_3);
      else if (object_class.find("TransformToPN") != string::npos)
            return HANDLER_D(TransformToPN:: handler_4);
      else if (object_class.find("Values") != string::npos)
            return HANDLER_D(Values:: handler_5);
      else
            return HANDLER_D(Compute:: handler_6);
}
```

5.6 Файл BaseClass.h

Листинг 4 – BaseClass.h

```
#ifndef BASECLASS_H
#define BASECLASS_H
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <typeinfo>
#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HANDLER_D(hendler_f) (TYPE_HANDLER) (&hendler_f)
using namespace std;

class BaseClass;
struct o_sh;

typedef void (BaseClass::*TYPE_SIGNAL)(string&, map<string, bool>&);
typedef void (BaseClass::*TYPE_HANDLER)(string, map<string, bool>);

class BaseClass{
private:
      struct o_sh{
            TYPE_SIGNAL p_signal;
            BaseClass* p_base;
            TYPE_HANDLER p_handler;
      };
      vector<o_sh*> connects;
      int ready = 1;
```

```
	string name;
	BaseClass * header;
public:
	vector<BaseClass *> subordinate_objects;
	BaseClass(BaseClass * head, string object_name = "root");
	virtual ~BaseClass();
	string get_name();
	void set_name(string new_name);
	void set_header(BaseClass *new_head);
	BaseClass * get_header();
	void print_tree(int);
	void print_tree_ready(int);
	BaseClass* get_object_by_name(string find_name);
	BaseClass* find_way(string, BaseClass*);
	string get_absolute_way();
	int get_ready();
	void set_ready(int);
	void set_connect(TYPE_SIGNAL, BaseClass*, TYPE_HANDLER);
	void delete_connect(TYPE_SIGNAL, BaseClass*, TYPE_HANDLER);
	void emit_signal(TYPE_SIGNAL);
	TYPE_SIGNAL get_signal(BaseClass*, bool = 1);
	TYPE_HANDLER get_handler(BaseClass*);
};

#endif
```

5.7 Файл Compute.cpp

Листинг 5 – Compute.cpp

```
#include "Compute.h"
#include <sstream>
#include <stack>

void Compute::signal_6(string& text, map<string, bool>& ){
	text = "Result: ";
	if (ans)
		text += "true";
	else
		text += "false";
}

void Compute::handler_6(string text, map<string, bool> values){
	if (text.size() > 0)
		pn = text;
	if (!values.empty())
		arguments = values;
	if (pn != "" && !arguments.empty()){
		stack<bool> current_vals;
		istringstream ss(pn + " ");
		string word;
		while(getline(ss, word, ' ')){
			if (word.size() == 0)
```

```
                        continue;
                  bool is_exist = 0;
                  for (auto &el: arguments){
                        if (el.first == word)
                              is_exist = 1;
                  }
                  if (is_exist)
                        current_vals.push(arguments[word]);
                  else{
                        bool a, b;
                        if (word != "NOT"){
                              b = current_vals.top();
                              current_vals.pop();
                              a = current_vals.top();
                              current_vals.pop();
                              if (word == "AND")
                                    current_vals.push(a * b);
                              else if (word == "OR")
                                    current_vals.push(a || b);
                              else if (word == "=>")
                                    current_vals.push(!a || b);
                              else if (word == "<=>")
                                    current_vals.push(a == b);
                              else
                                    current_vals.push(a ^ b);
                        }
                        else{
                              a = current_vals.top();
                              current_vals.pop();
                              current_vals.push(!a);
                        }
                  }
            }
            ans = current_vals.top();
      }
}
```

5.8 Файл Compute.h

Листинг 6 – Compute.h

```
#ifndef COMPUTE_H
#define COMPUTE_H
#include "BaseClass.h"

class Compute: public BaseClass{
private:
      bool ans;
      string pn;
      map<string, bool> arguments;
public:
      using BaseClass::BaseClass;
      void signal_6(string&, map<string, bool>&);
```

```
	void handler_6(string, map<string, bool>);
};

#endif
```

5.9 Файл main.cpp

Листинг 7 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "Application.h"
using namespace std;

int main(){
	Application object(nullptr);
	object.build_tree_objects();
	return object.exec_app();
}
```

5.10 Файл Printer.cpp

Листинг 8 – Printer.cpp

```
#include "Printer.h"

void Printer::handler_3(string text, map<string, bool>){
	cout << "\n" << text;
}
```

5.11 Файл Printer.h

Листинг 9 – Printer.h

```
#ifndef PRINTER_H
#define PRINTER_H
#include "BaseClass.h"

class Printer: public BaseClass{
public:
	using BaseClass::BaseClass;
	void handler_3(string, map<string, bool>);
};
```

```
#endif
```

5.12 Файл Reader.cpp

Листинг 10 – Reader.cpp

```
#include "Reader.h"

void Reader::signal_pn_2(string& text, map<string, bool>&){
	getline(cin, text);
}

void Reader::signal_values_2(string& text, map<string, bool>&){
	getline(cin, text);
}
```

5.13 Файл Reader.h

Листинг 11 – Reader.h

```
#ifndef READER_H
#define READER_H
#include "BaseClass.h"

class Reader: public BaseClass{
public:
	using BaseClass::BaseClass;
	void signal_pn_2(string&, map<string, bool>&);
	void signal_values_2(string&, map<string, bool>&);
};

#endif
```

5.14 Файл TransformToPN.cpp

Листинг 12 – TransformToPN.cpp

```
#include "TransformToPN.h"
#include <stack>
#include <sstream>

void TransformToPN::signal_send_4(string& text, map<string, bool>&){
	text = pn;
}
```

```
void TransformToPN::signal_print_4(string& text, map<string, bool>&){
	text = "Polish Notation:" + pn;
}

void TransformToPN::handler_4(string text, map<string, bool>){
	string word;
	stack<string> current_vals;
	istringstream formula(text.substr(0, text.size() - 1));
	map<string, int> actions = {{"(", 1}, {")", 1}, {"<=>", 2}, {"=>", 3},
{"XOR", 4}, {"OR", 4}, {"AND", 5}, {"NOT", 6} };
	pn = "";
	while (getline(formula, word, ' ')){
		if (word.size() == 0)
			continue;
		bool is_exist = 0;
		for (auto &el: actions){
			if (el.first == word)
				is_exist = 1;
		}
		if (!is_exist)
			pn += " " + word;
		else{
			if (word == "(")
				current_vals.push(word);
			else if (word == ")"){
				while (!current_vals.empty() && current_vals.top() != "(")
{
					pn += " " + current_vals.top();
					current_vals.pop();
				}
				current_vals.pop();
			}
			else{
				while (!current_vals.empty() &&
actions[current_vals.top()] >= actions[word]){
					pn += " " + current_vals.top();
					current_vals.pop();
				}
				current_vals.push(word);
			}
		}
	}
	while (!current_vals.empty()){
		pn += " " + current_vals.top();
		current_vals.pop();
	}
}
```

5.15 Файл TransformToPN.h

Листинг 13 – TransformToPN.h

```
#ifndef TRANSFORMTOPN_H
```

```
#define TRANSFORMTOPN_H
#include "BaseClass.h"

class TransformToPN: public BaseClass{
private:
	string pn;
public:
	using BaseClass::BaseClass;
	void signal_send_4(string&, map<string, bool>&);
	void signal_print_4(string&, map<string, bool>&);
	void handler_4(string, map<string, bool>);
};

#endif
```

5.16 Файл Values.cpp

Листинг 14 – Values.cpp

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

void Values::signal_send_5(string& text, map<string, bool>& values){
	values = arguments;
}

void Values::signal_print_5(string& text, map<string, bool>& values){
	text = "Values:";
	for (auto &el: arguments){
		if (el.second)
			text += " " + el.first + " = true";
		else
			text += " " + el.first + " = false";
	}
}

void Values::handler_5(string text, map<string, bool>){
	arguments = {};
	istringstream ss(text);
	string word, name;
	bool value;
	int tmp = 0;
	while (getline(ss, word, ' ')){
		tmp++;
		if (tmp == 1)
			name = word;
		else if (tmp == 3){
			if (word == "0")
				value = 0;
			else
				value = 1;
			arguments[name] = value;
			tmp = 0;
```

```
            }
      }
}
```

5.17 Файл Values.h

Листинг 15 – Values.h

```
#ifndef VALUES_H
#define VALUES_H
#include "BaseClass.h"

class Values: public BaseClass{
private:
      map<string, bool> arguments;
public:
      using BaseClass::BaseClass;
      void signal_send_5(string&, map<string, bool>&);
      void signal_print_5(string&, map<string, bool>&);
      void handler_5(string, map<string, bool>);
};

#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 18.

Таблица 18 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 0 b = 1 c = 1 .	OUT Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true	OUT Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true
SHOWTREE	OUT root is ready Reader is ready Printer is ready Values is ready TransformToPN is ready Compute is ready	OUT root is ready Reader is ready Printer is ready Values is ready TransformToPN is ready Compute is ready
.	OUT	OUT
c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 0 b = 1 c = 1 c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 1 b = 0 c = 0 .	OUT Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = true b = false c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: false	OUT Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = true b = false c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: false
c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 0 b = 1 c = 1 SHOWTREE	OUT Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true root is ready Reader is ready Printer is ready Values is ready TransformToPN is ready Compute is ready	OUT Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true root is ready Reader is ready Printer is ready Values is ready TransformToPN is ready Compute is ready

ЗАКЛЮЧЕНИЕ

Во время выполнения работы были достигнуты следующие цели:

Освоение объектно-ориентированногго програмирования

Освоение объектно-ориентированного языка программирования C++

Закрепление навыка разработки программы

Освоение выполнения всех необходимых работ согласно этапам разработки программы и соответствующих программных инструментов

Освоение умения разработки программы как системы

Освоение умения проектирования архитектуры программы на базе построения иерархии объектов

Освоение навыка программирования по заранее определенным правилам

Освоение весионности при разраотке программ

Использование единой базы данных

Тестирование работы программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. 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).

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).