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].

Цель курсовой работы: моделирование работы логического калькулятора, используя сигналы и обработчики. Для достижения поставленной цели необходимо выполнить следующие задачи:

- Освоение принципов объектно-ориентированного программирования;
- Освоение основ объектно-ориентированного программирования;
- Освоение умения проектирования архитектуры программы на базе построения иерархии объектов;
- Освоение выполнения всех необходимых работ согласно этапам разработки программы и соответствующих программных инструментов;
- Моделироание работы логического калькулятора при помощи сигналов и обработчиков;
- Построение системы взаимодействия объектов при помощи интерфейса сигналов и обработчиков;
- Описание алгоритма работы программы;
- Построение блок-схем для разработанного алгоритма;
- Написание кода на языке программирования C++, согласно разработанному алгоритму работы программы;
- 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. Вывод осуществляется в алфавитном порядке идентификаторов 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.

Объект "система" помимо построения дерева иерархии объектов и запуска основного алгоритма работы системы может выполнять и другие функции.

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 обратной 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»…
Вывод осуществляется в алфавитном порядке идентификаторов переменных.

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 задачи используется:

- объект input класса cl_input предназначен для ввод строки функции и переменных;
- объект form класса cl_form предназначен для обработка введённых переменных и формирование множества ;
- объект rpn класса cl_rpn предназначен для обработка введённой строки функции и формирование обратной польской нотации;
- объект calc класса cl_calc предназначен для вычисление значения функци;
- объект print класса cl_print предназначен для вывод обработки результатов измерений;
- объект ob_cl_application класса cl_application предназначен для корневой объект системы;
- map - упорядоченый ассоциативный контейнер стандартной библиотеки (словарь);
- stack - последовательный контейнер с доступом к вершинному элементу ;
- stringstream - класс оперирования потоком строк;
- tYPE_SIGNAL - Тип сигнала изменён на дополнительный параметр типа map<string, bool>;
- tYPE_HANDLER - Тип обработчика изменён на дополнительный параметр типа map<string, bool>;
- emit_signal - Метод инициализации сигнала изменён на получение и отправку дополнительного параметра типа map<string, bool>.

Класс cl_input:

- функционал:
    - метод signal_form_vars — метод сигнала на формирование

множества переменных;

- метод signal_form_rpn — метод сигнала на формирование обратной польской нотации;
- метод handler_input_expr — метод ввода строки функции;
- метод handler_input_vars — метод ввода строки переменых;
- метод cl_input — параметризированный конструктор.

Класс cl_form:

- функционал:
  - метод signal_pass_vars — сигнал отправки сформированных переменных;
  - метод handler_form_vars — метод формирования множества переменных;
  - метод cl_form — параметризированный конструктор.

Класс cl_rpn:

- функционал:
  - метод signal_pass_rpn — сигнал передачи сформированной обратной польской нотации;
  - метод handler_form_rpn — метод формирования обратной польской нотации;
  - метод isOp — метод проверки символа на операцию;
  - метод cl_rpn — параметризированный конструктор.

Класс cl_calc:

- свойства/поля:
  - поле обратная польская нотация:
    - наименование — rpn;
    - тип — string;
    - модификатор доступа — private;

- поле сформированнон множество переменных:
  - наименование — vars;
  - тип — map<string, bool>;
  - модификатор доступа — private;

- функционал:
  - метод signal_pass_result — сигнал передачи результата вычислений;
  - метод handler_get_rpn — метод сохранения обратной польской нотации;
  - метод handler_get_vars — метод сохранения множества переменных;
  - метод calculate — метод обработки сохранённых значений, только при получении всех параметров;
  - метод operate — метод выполнения одной операции;
  - метод isOp — метод проверки символа на операцию;
  - метод cl_calc — параметризированный конструктор.

Класс cl_print:

- свойства/поля:
  - поле rpn:
    - наименование — Обратная польская нотация;
    - тип — string;
    - модификатор доступа — private;
  - поле vars:
    - наименование — Сформированное множество переменных;
    - тип — string;
    - модификатор доступа — private;
  - поле result:
    - наименование — Результат измерений;
    - тип — string;

        - модификатор доступа — private;
- функционал:
    - метод handler_get_vars — метод сохранения множества переменных;
    - метод handler_get_rpn — метод сохранения обратной польской нотации;
    - метод handler_get_result — метод сохранения результата;
    - метод print_all — метод вывода сохранённых результатов, только при получении всех параметров;
    - метод cl_print — параметризированный конструктор.

Класс cl_application:

- функционал:
    - метод signal_input_expr — сигнал на ввод строки функции;
    - метод signal_input_vars — сигнал на ввод переменных;
    - метод build_tree_objects — метод построения дерева объектов;
    - метод exec_app — метод запуска системы.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_input			Класс оперирующий вводом строки функции и переменных	
2	cl_form			Класс формирующий множество перменных из введённых	
3	cl_rpn			Класс формирующий обратную польскую нотацию из введённой формулы	
4	cl_calc			Класс для вычисления значения функции	
5	cl_print			Класс для вывода обработки результатов измерений	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
6	cl_application			Корневой объект системы	

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 signal_form_vars класса cl_input

Функционал: Метод сигнала на формирование множества переменных.

Параметры: string& - ссылка на сообщение map<string, bool>& - ссылка на множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода signal_form_vars класса cl_input

№	Предикат	Действия	№ перехода
1		0	∅

3.2 Алгоритм метода signal_form_rpn класса cl_input

Функционал: Метод сигнала на формирование обратной польской нотации.

Параметры: string& - ссылка на сообщение map<string, bool>& - ссылка на множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода signal_form_rpn класса cl_input

№	Предикат	Действия	№ перехода
1		0	∅

3.3 Алгоритм метода handler_input_expr класса cl_input

Функционал: Метод ввода строки функции.

Параметры: string - сообщение map<string, bool> - множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода handler_input_expr класса cl_input

№	Предикат	Действия	№ перехода
1		Объявление переменной expression типа string	2
2		Ввод значения expression из строки потокового ввода	3
3	expression == "."	Завершение программы с кодом 0	∅
	expression == "SHOWTREE"	Вызов метода print_state у объекта по указателю header_ptr	4
		Инициирование сигнала методом emit_signal: сигнал signal_form_rpn класса cl_input с передачей expresion и пустого словаря типа map<string, bool>	∅
4		Завершение программы с кодом 0	∅

3.4 Алгоритм метода handler_input_vars класса cl_input

Функционал: Метод ввода строки переменых.

Параметры: string - сообщение map<string, bool> - множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода handler_input_vars класса cl_input

№	Предикат	Действия	№ перехода
1		Объявление переменной vars типа стринг	2

№	Предикат	Действия	№ перехода
2		Ввод значения vars из строки потокового ввода	3
3		Инициирование сигнала методом emit_signal: сигнал signal_form_vars класса cl_input с передачей vars и пустого словаря типа map<string, bool>	∅

3.5 Алгоритм метода signal_pass_vars класса cl_form

Функционал: Сигнал отправки сформированных переменных.

Параметры: string& - ссылка на сообщение map<string, bool>& - ссылка на множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода signal_pass_vars класса cl_form

№	Предикат	Действия	№ перехода
1		0	∅

3.6 Алгоритм метода handler_form_vars класса cl_form

Функционал: Метод формирования множества переменных.

Параметры: string vars - строка переменных map<string, bool> - множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода handler_form_vars класса cl_form

№	Предикат	Действия	№ перехода
1		Объявление переменной ss строкового потока	2

№	Предикат	Действия	№ перехода
		stringstream	
2		Ввод vars в поток ss	3
3		Объявление переменной словаря formed_vars типа map<string, bool>	4
4	Не конец потока ss	Объявление переменных symbol, eql, val типа string	5
		Инициирование сигнала методом emit_signal: сигнал signal_pass_vars класса cl_form с передачей пустой строки и словаря formed_vars	∅
5		Ввод значений symbol, eql, val из потока ss	6
6		Объявление переменной v типа bool	7
7	val == "1"	v = true	8
		v = false	8
8		Вставка в словарь formed_vars по ключу symbol значение v при помощи метода словаря insert	4

3.7 Алгоритм метода signal_pass_rpn класса cl_rpn

Функционал: Сигнал передачи сфорированной обратной польской нотации.

Параметры: string& - ссылка на сообщение map<string, bool>& - ссылка на множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода signal_pass_rpn класса cl_rpn

№	Предикат	Действия	№ перехода
1		0	∅

3.8 Алгоритм метода handler_form_rpn класса cl_rpn

Функционал: Метод формирования обратной польской нотации.

Параметры: string expression - строка функции map<string, bool> - множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода handler_form_rpn класса cl_rpn

№	Предикат	Действия	№ перехода
1		Инициализация переменной priority_map - словаря приоритетов типа map<string, int> словарём {"NOT": 5, "AND": 4, "XOR": 3, "OR": 2, "=>": 1, "<=>": 0};	2
2		Объявление переменной ss строкового потока stringstream	3
3		Ввод expression в поток ss	4
4		Объявление переменной стека stc типа stack<string>	5
5		Инициализация переменной result типа string пустой строкой	6
6	Не конец потока ss		7
			20
7		Объявление переменной l типа string	8
8		Ввод значения l из потока ss	9
9	В l есть символ '.'	Удаление из l последнего символа вызовом метода строки pop_back	10
			10
10	Значение вызова метода isOp с параметром l ложь и l != "("	Добавление в конец result значение l	6

№	Предикат	Действия	№ перехода
	и l != ")"		
	l == "NOT" или l == "("	Добавление l в конец стека stc вызовом метода стека stc push с передачей параметра l	6
	l == ")"		11
			14
11	Значение вызова метода top объекта stc != "("	Добавление в конец result значение вызова метода top объекта stc	12
			13
12		Вызов метода pop объекта stc	11
13		Вызов метода pop объекта stc	6
14	Значение вызова метода isOp с параметром l истина		15
			6
15	stc не пустой		16
			19
16	Вершина стека stc == "NOT" или значение priority_map по ключу значения вершины стека >= значение priority_map по ключу l	Добавление в конец result значение вызова метода top объекта stc	17
			19
17		Вызов метода pop объекта stc	18
18	stc пустой		19
			16
19		Добавление l в конец стека stc вызовом метода стека stc push с передачей параметра l	6
20	stc не пустой	Добавление в конец result значение вызова метода top объекта stc	21
			22

№	Предикат	Действия	№ перехода
21		Вызов метода pop объекта stc	20
22		Инициирование сигнала методом emit_signal: сигнал signal_pass_rpn класса cl_rpn с передачей result и пустого словаря типа map<string, bool>	∅

3.9 Алгоритм конструктора класса cl_input

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_input

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора родительского класса cl_base с передачей параметров header и name	∅

3.10 Алгоритм конструктора класса cl_form

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_form

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора родительского класса cl_base с передачей параметров header и name	∅

3.11 Алгоритм метода isOp класса cl_rpn

Функционал: Метод проверки символа на операцию.

Параметры: string op - проверяемый символ.

Возвращаемое значение: bool - результат проверки.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода isOp класса cl_rpn

№	Предикат	Действия	№ перехода
1		Возврат op == "NOT" или op == "AND" или op == "XOR" или op == "OR" или op == "<=>" или op == "=>"	∅

3.12 Алгоритм конструктора класса cl_rpn

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса cl_rpn

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора родительского класса cl_base с передачей параметров header и name	∅

3.13 Алгоритм метода signal_pass_result класса cl_calc

Функционал: Сигнал передачи результата вычислений.

Параметры: string& - ссылка на сообщение map<string, bool>& - ссылка на множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода signal_pass_result класса cl_calc

№	Предикат	Действия	№ перехода
1		0	∅

3.14 Алгоритм метода handler_get_rpn класса cl_calc

Функционал: Метод сохранения обратной польской нотации.

Параметры: string rpn - обратная польская запись map<string, bool> - множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода handler_get_rpn класса cl_calc

№	Предикат	Действия	№ перехода
1		Значение поле rpn = аргумент rpn	2
2		Вызов метода calculate текущего объекта	∅

3.15 Алгоритм метода handler_get_vars класса cl_calc

Функционал: Метод сохранения множества переменных.

Параметры: string - сообщение, map<string, bool> vars - сформированное множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода handler_get_vars класса cl_calc

№	Предикат	Действия	№ перехода
1		Значение поле vars = аргумент vars	2

№	Предикат	Действия	№ перехода
2		Вызов метода calculate текущего объекта	∅

3.16 Алгоритм метода calculate класса cl_calc

Функционал: Метод обработки сохранённых значений, только при получении всех параметров.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода calculate класса cl_calc

№	Предикат	Действия	№ перехода
1	Поле rpn == "" или поле словарь vars пустой	Возврат	∅
		Объявление переменной ss строкового потока stringstream	2
2		Ввод rpn в поток ss	3
3		Объявление переменной стека stc типа stack<bool>	4
4	Не конец потока ss	Объявление переменной l типа string	5
			14
5		Объявление переменных a, b типа bool	6
6		Ввод значения l из потока ss	7
7	Значение вызова метода isOp с параметром l ложь	Вызов метода стека push у объекта stc с передачей аргумента значение поля в словаре vars по ключу l	4
	l == "NOT"	a = значение вызова метода top объекта stc	8
		a = значение вызова метода top объекта stc	10
8		Вызов метода pop объекта stc	9
9		Вызов метода стека push у объекта stc с передачей аргумента значение !a	4

№	Предикат	Действия	№ перехода
10		Вызов метода pop объекта stc	11
11		b = значение вызова метода top объекта stc	12
12		Вызов метода pop объекта stc	13
13		Вызов метода стека push у объекта stc с передачей аргумента значение вызова метода operate текущего объекта с передачей аргументов значений b, a, l	4
14		Объявление переменной result типа string	15
15	Значение вызова метода top объекта stc истина	result = "true"	16
		result = "false"	16
16		rpn = ""	17
17		vars = {}	18
18		Инициирование сигнала методом emit_signal: сигнал signal_pass_result класса cl_calc с передачей result и пустого словаря типа map<string, bool>	∅

3.17 Алгоритм метода operate класса cl_calc

Функционал: Метод выполнения одной операции.

Параметры: bool a, bool b - операнды, string op - оператор.

Возвращаемое значение: bool - результат операции.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода operate класса cl_calc

№	Предикат	Действия	№ перехода
1	op == "AND"	Возврат a && b	∅
	op == "XOR"	Возврат a != b	∅
	op == "OR	Возврат a || b	∅

№	Предикат	Действия	№ перехода
	op == "=>"	Возврат !a || b	∅
	op == "<=>	Возврат a == b	∅
		Возврат a	∅

3.18 Алгоритм метода isOp класса cl_calc

Функционал: Метод проверки символа на операцию.

Параметры: string op - проверяемый символ.

Возвращаемое значение: bool - результат проверки.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода isOp класса cl_calc

№	Предикат	Действия	№ перехода
1		Возврат op == "NOT" или op == "AND" или op == "XOR" или op == "OR" или op == "<=>" или op == "=>"	∅

3.19 Алгоритм конструктора класса cl_calc

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 20.

Таблица 20 – Алгоритм конструктора класса cl_calc

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора родительского класса cl_base с передачей параметров header и name	∅

3.20 Алгоритм метода handler_get_rpn класса cl_print

Функционал: Метод сохранения множества переменных.

Параметры: string rpn - обратная польская запись, map<string, bool> vars - множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода handler_get_rpn класса cl_print

№	Предикат	Действия	№ перехода
1		Значение поля rpn = аргумент rpn	2
2		Вызов метода print_all текущего объекта	∅

3.21 Алгоритм метода handler_get_vars класса cl_print

Функционал: Метод сохранения обратной польской нотации.

Параметры: string - сообщение, map<string, bool> formed_vars - сформированное множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 22.

Таблица 22 – Алгоритм метода handler_get_vars класса cl_print

№	Предикат	Действия	№ перехода
1		Объявление интератора it по словарю formed_vars	2
2	it != концу formed_vars		3
			8
3	it != началу formed_vars	vars += " "	4
			4
4		Объявление переменной val типа string	5
5	Значение по ключу первого	val = "true"	6

№	Предикат	Действия	№ перехода
	значенения в it в словаре formed_vars истина		
		val = "false"	6
6		vars += первое значение в it + " = " + val	7
7		Шаг итерации it	2
8		Вызов метода print_all текущего объекта	∅

3.22 Алгоритм метода handler_get_result класса cl_print

Функционал: Метод сохранения результата.

Параметры: string result - результат вычислений, map<string, bool> - множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода handler_get_result класса cl_print

№	Предикат	Действия	№ перехода
1		Значение поля result = аргумент result	2
2		Вызов метода print_all текущего объекта	∅

3.23 Алгоритм метода print_all класса cl_print

Функционал: Метод вывода сохранённых результатов, только при получении всех параметров.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода print_all класса cl_print

№	Предикат	Действия	№ перехода
1	rpn == "" или vars == "" или result == ""	Возврат	∅
			2
2		Вывод "Values: " + значение поля vars	3
3		Вывод "Polish Notation: " + значение поля rpn	4
4		Вывод "Result: " + значение поля result	5
5		rpn = ""	6
6		vars = ""	7
7		result = ""	∅

3.24 Алгоритм конструктора класса cl_print

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - имя объекта.

Алгоритм конструктора представлен в таблице 25.

Таблица 25 – Алгоритм конструктора класса cl_print

№	Предикат	Действия	№ перехода
1		Вызов параметризированного конструктора родительского класса cl_base с передачей параметров header и name	∅

3.25 Алгоритм метода signal_input_expr класса cl_application

Функционал: Сигнал на ввод строки функции.

Параметры: string& - ссылка на сообщение map<string, bool>& - ссылка на множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода signal_input_expr класса cl_application

№	Предикат	Действия	№ перехода
1		0	∅

3.26 Алгоритм метода signal_input_vars класса cl_application

Функционал: Сигнал на ввод переменных.

Параметры: string& - ссылка на сообщение map<string, bool>& - ссылка на множество переменных .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода signal_input_vars класса cl_application

№	Предикат	Действия	№ перехода
1		0	∅

3.27 Алгоритм метода build_tree_objects класса cl_application

Функционал: Метод построения дерева объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода edit_name текущего объекта с передачей параметра "Logical Calculator"	2
2		Инициализация указателя input на объект типа cl_base выделением	3

№	Предикат	Действия	№ перехода
		памяти объекту класса cl_input с передачей параметров конструктору указатель на текущий объект и "input"	
3		Инициализация указателя form на объект типа cl_base выделением памяти объекту класса cl_form с передачей параметров конструктору указатель на текущий объект и "form"	4
4		Инициализация указателя rpn на объект типа cl_base выделением памяти объекту класса cl_rpn с передачей параметров конструктору указатель на текущий объект и "rpn"	5
5		Инициализация указателя print на объект типа cl_base выделением памяти объекту класса cl_calc с передачей параметров конструктору указатель на текущий объект и "calc"	6
6		Инициализация указателя calc на объект типа cl_base выделением памяти объекту класса cl_print с передачей параметров конструктору указатель на текущий объект и "print"	7
7		Установка связи вызовом метода set_connect у текущего объекта: Сигнал signal_input_expr класса cl_application к обработчику handler_input_expr класса cl_input объекта input	8
8		Установка связи вызовом метода set_connect у текущего объекта: Сигнал signal_input_vars класса cl_application к обработчику handler_input_vars класса cl_input объекта input	9
9		Установка связи вызовом метода set_connect у объекта input: Сигнал signal_form_vars класса cl_input к обработчику handler_form_vars класса cl_form объекта form	10
10		Установка связи вызовом метода set_connect у объекта input: Сигнал signal_form_rpn класса cl_input к обработчику handler_form_rpn класса cl_rpn объекта rpn	11
11		Установка связи вызовом метода set_connect у объекта form: Сигнал signal_pass_vars класса cl_form к обработчику handler_get_vars класса cl_calc объекта calc	12

№	Предикат	Действия	№ перехода
12		Установка связи вызовом метода set_connect у объекта form: Сигнал signal_pass_vars класса cl_form к обработчику handler_get_vars класса cl_print объекта print	13
13		Установка связи вызовом метода set_connect у объекта rpn: Сигнал signal_pass_rpn класса cl_rpn к обработчику handler_get_rpn класса cl_calc объекта calc	14
14		Установка связи вызовом метода set_connect у объекта rpn: Сигнал signal_pass_rpn класса cl_rpn к обработчику handler_get_rpn класса cl_print объекта print	15
15		Установка связи вызовом метода set_connect у объекта calc: Сигнал signal_pass_result класса cl_calc к обработчику handler_get_result класса cl_print объекта print	∅

3.28 Алгоритм метода exec_app класса cl_application

Функционал: Метод запуска системы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "OUT"	2
2		Инициализация указателя input на объект типа cl_base равному значению вызова метода текущего объекта get_child с передачей параметра "input"	3
3	Правда		4
		Возврат 0	∅

№	Предикат	Действия	№ перехода
4		Инициирование сигнала методом emit_signal: сигнал signal_input_expr класса cl_application с передачей пустых параметров	5
5		Инициирование сигнала методом emit_signal: сигнал signal_input_vars класса cl_application с передачей пустых параметров	3

3.29 Алгоритм функции main

Функционал: Главная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 30.

Таблица 30 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		<...> Повторение кода из КВ-4	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-14.

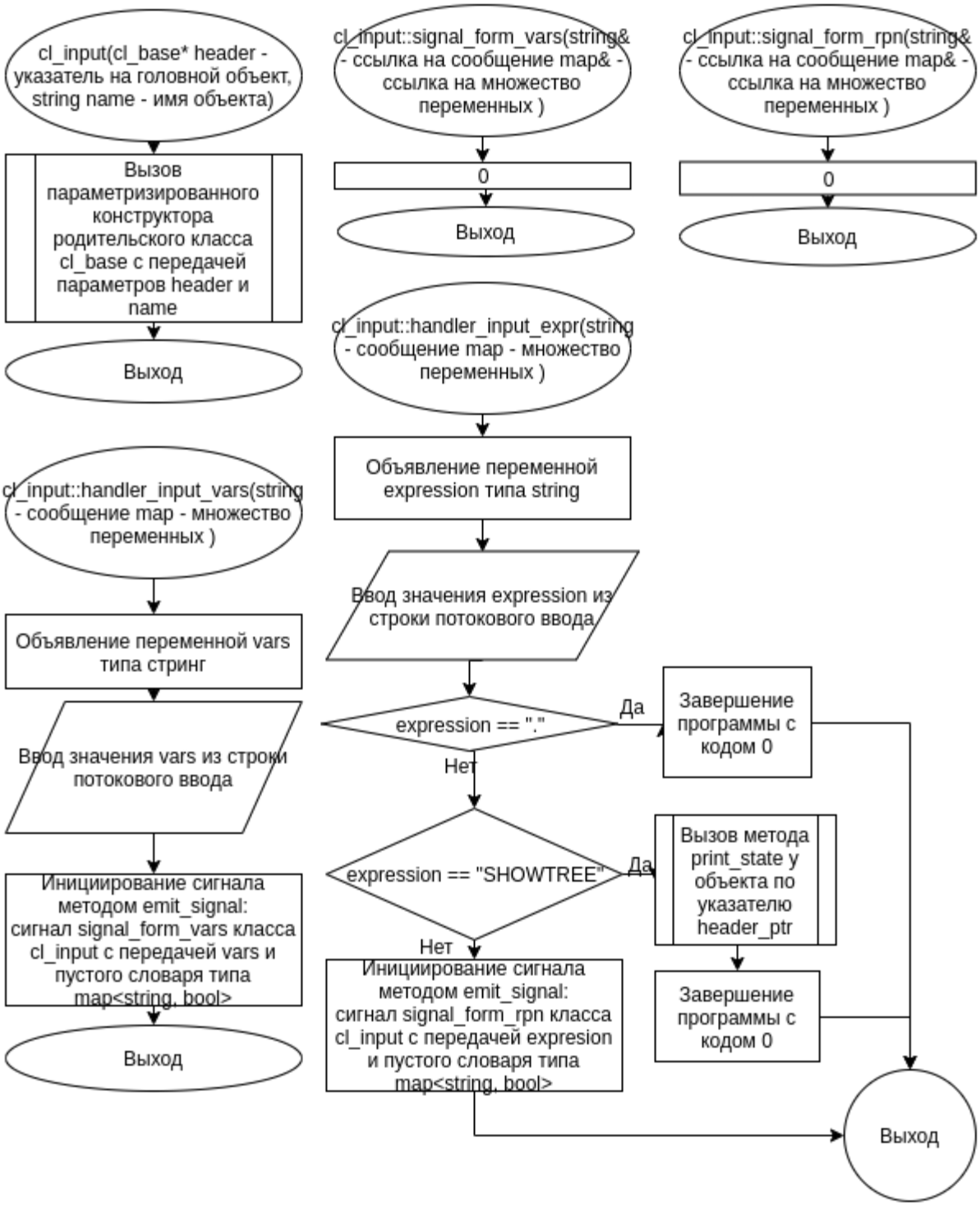

Рисунок 1 – Блок-схема алгоритма

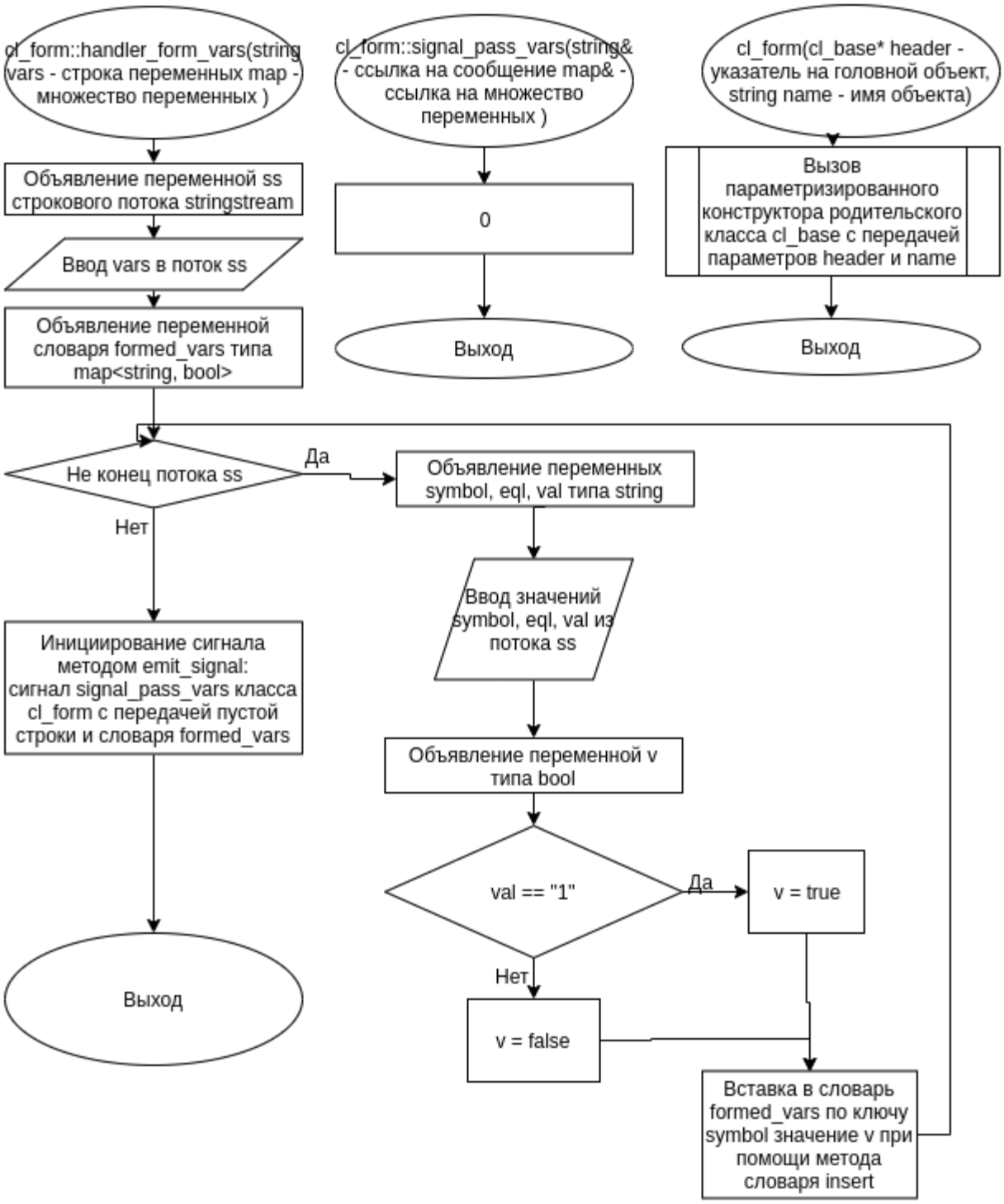

Рисунок 2 – Блок-схема алгоритма

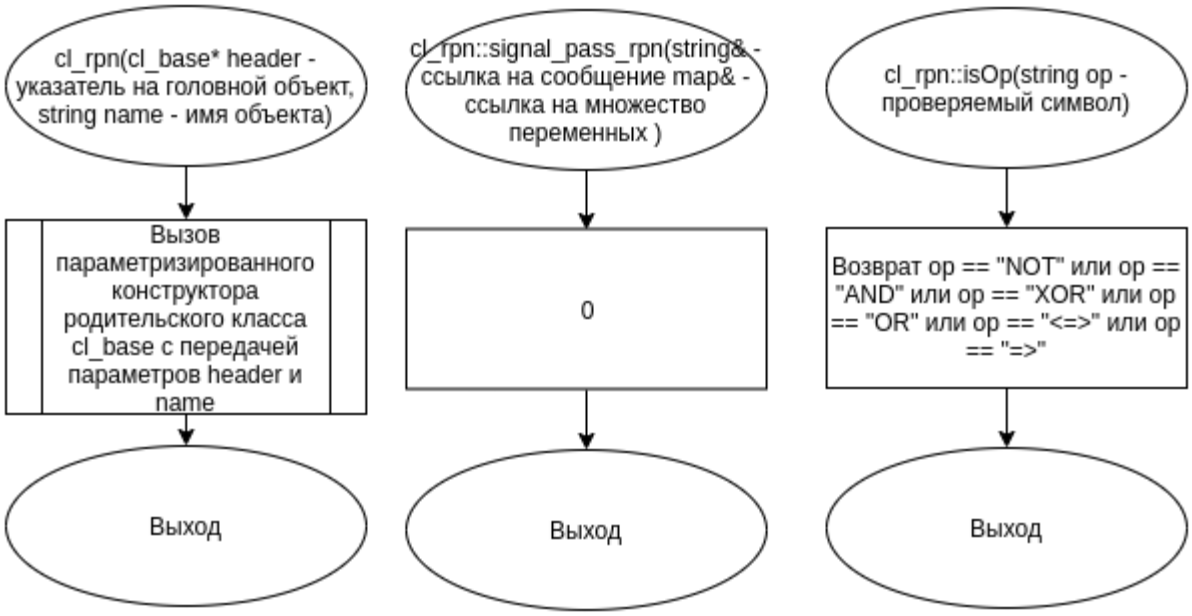

Рисунок 3 – Блок-схема алгоритма

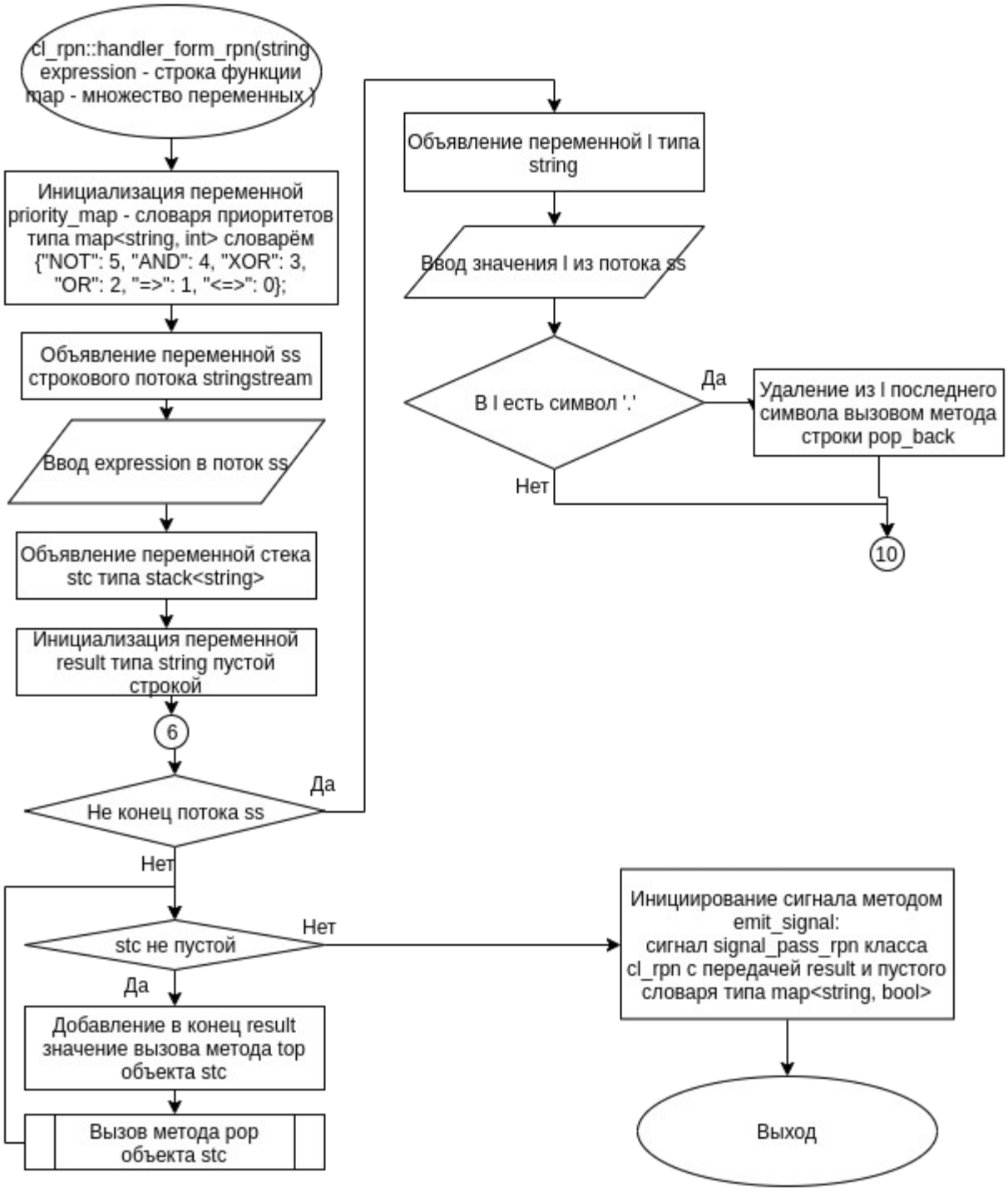

Рисунок 4 – Блок-схема алгоритма

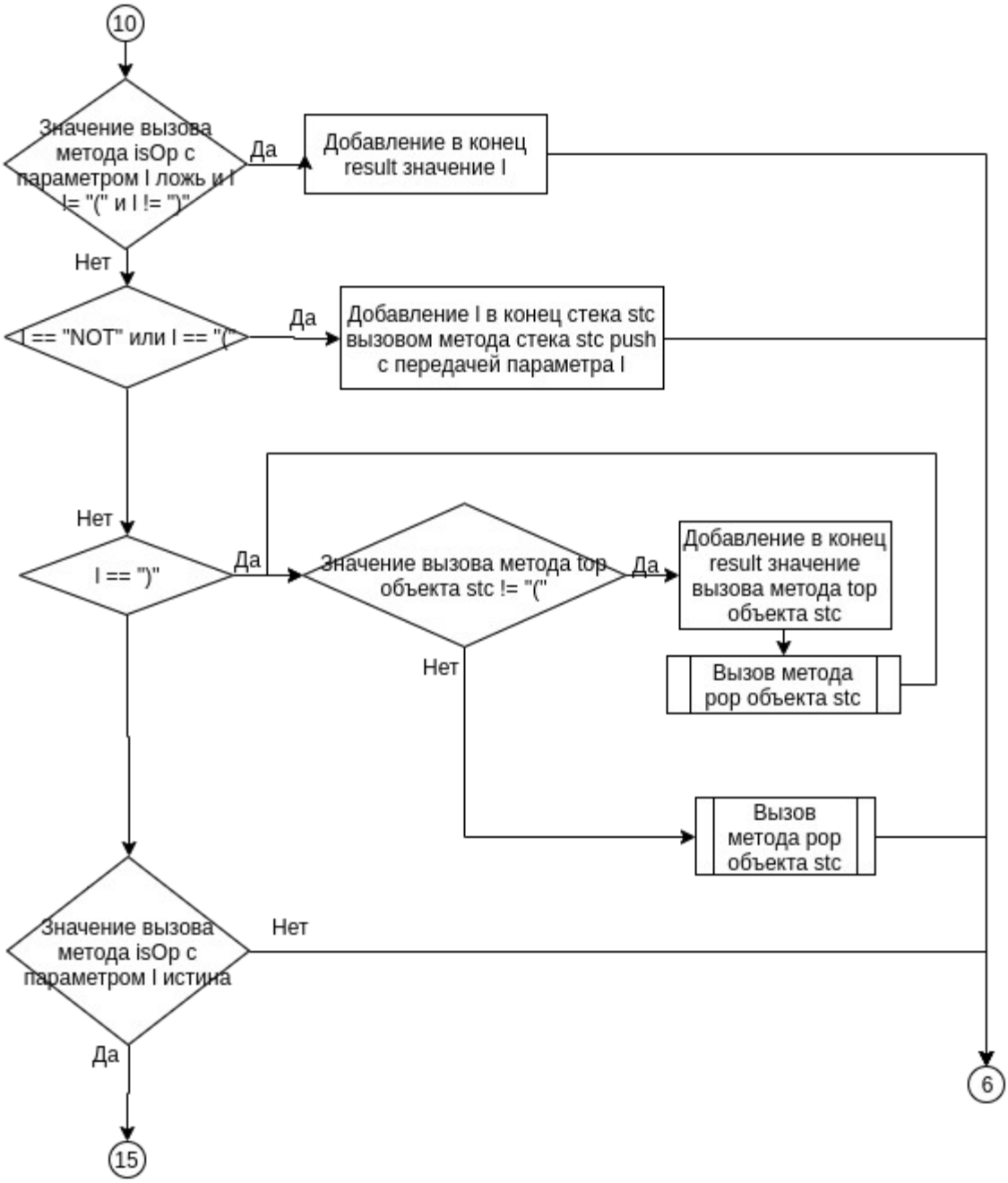

Рисунок 5 – Блок-схема алгоритма

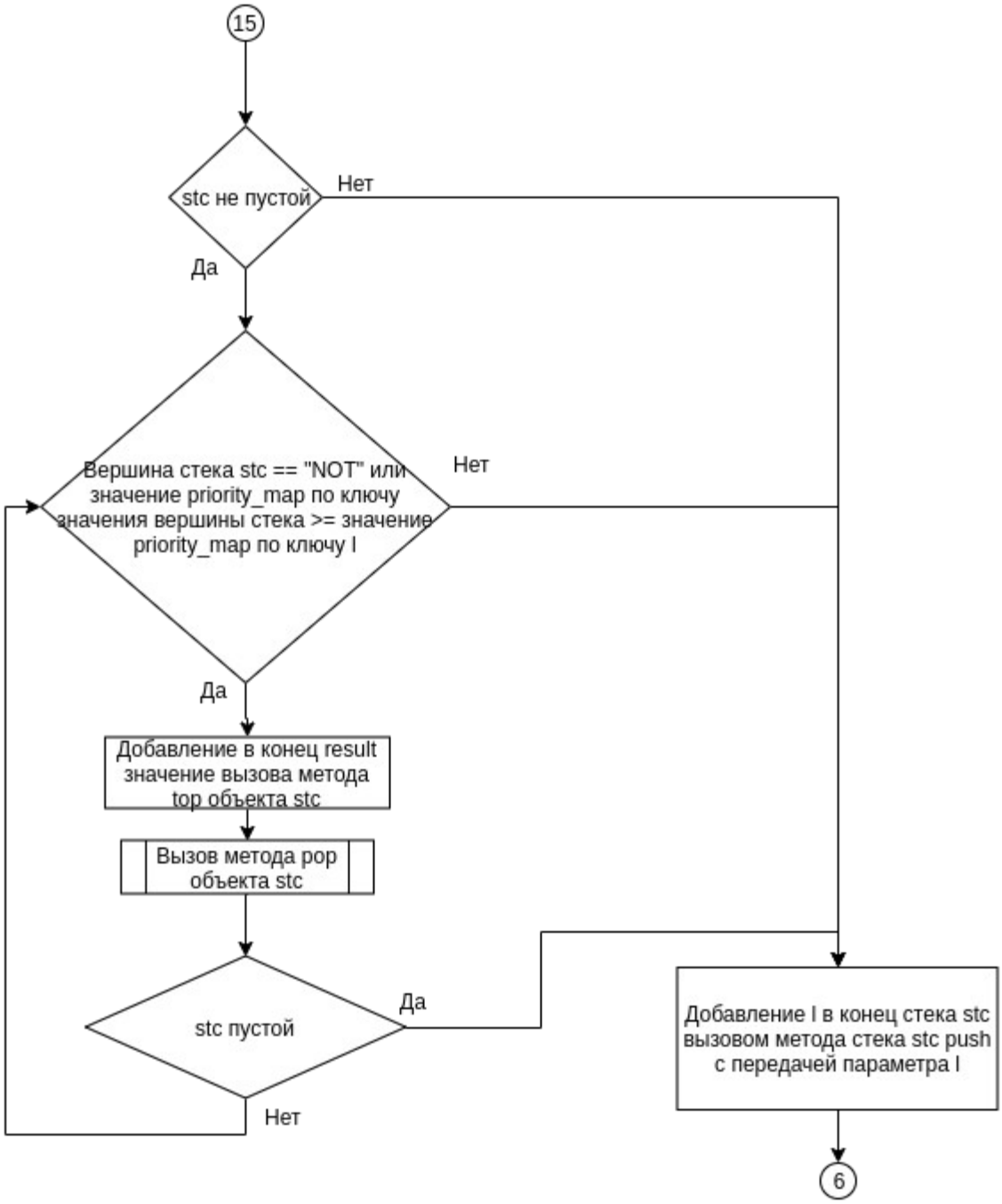

Рисунок 6 – Блок-схема алгоритма

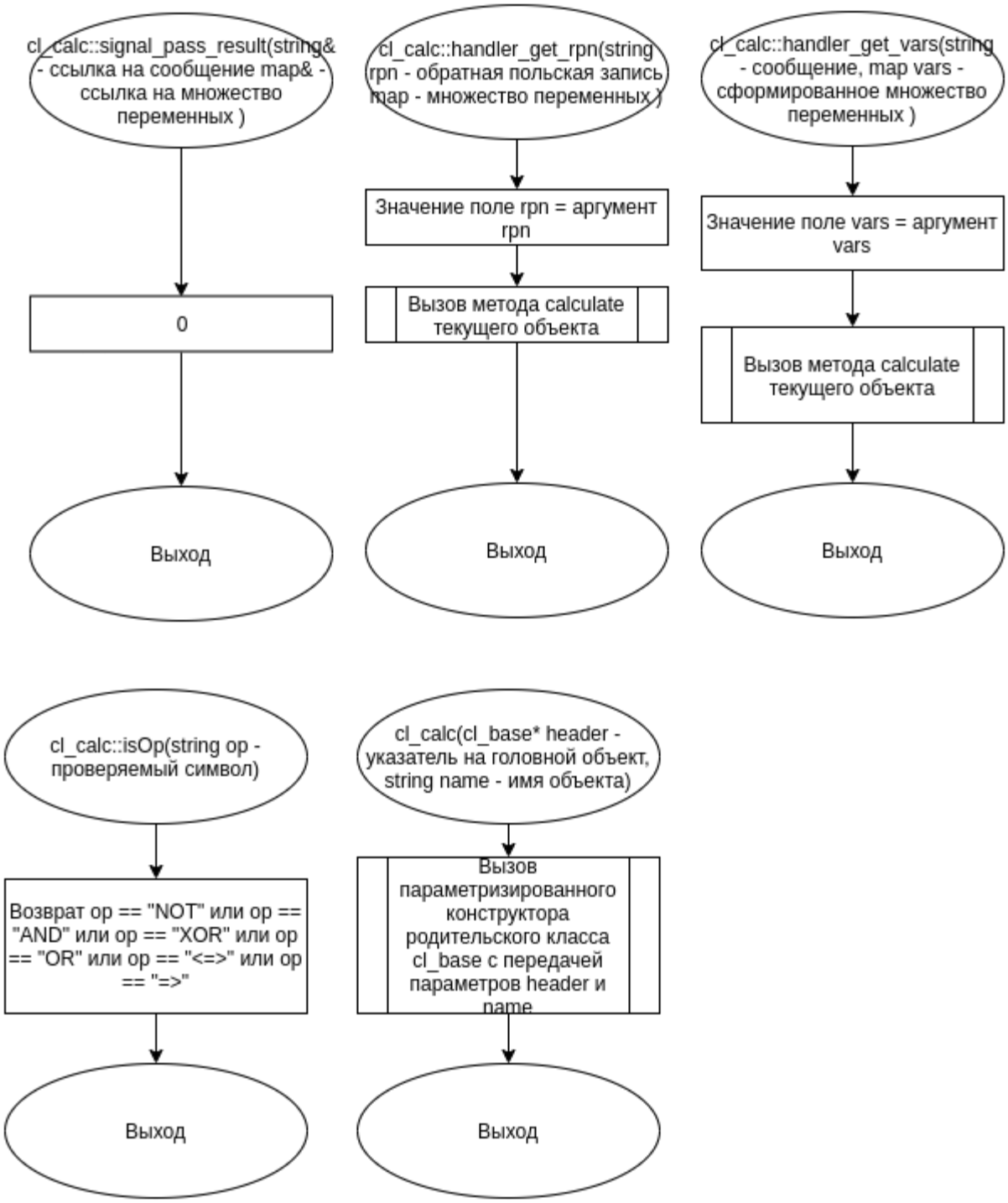

Рисунок 7 – Блок-схема алгоритма

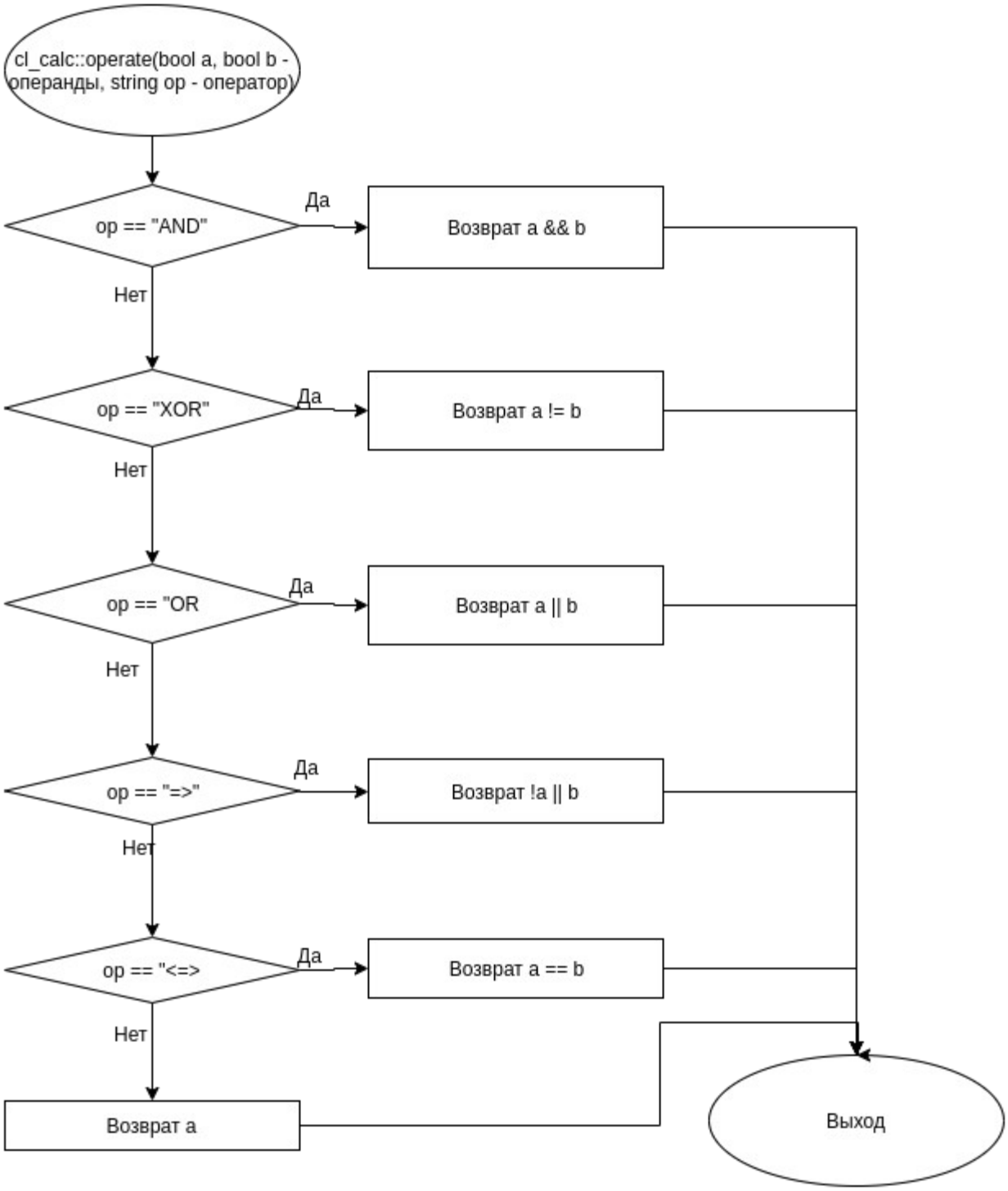

Рисунок 8 – Блок-схема алгоритма

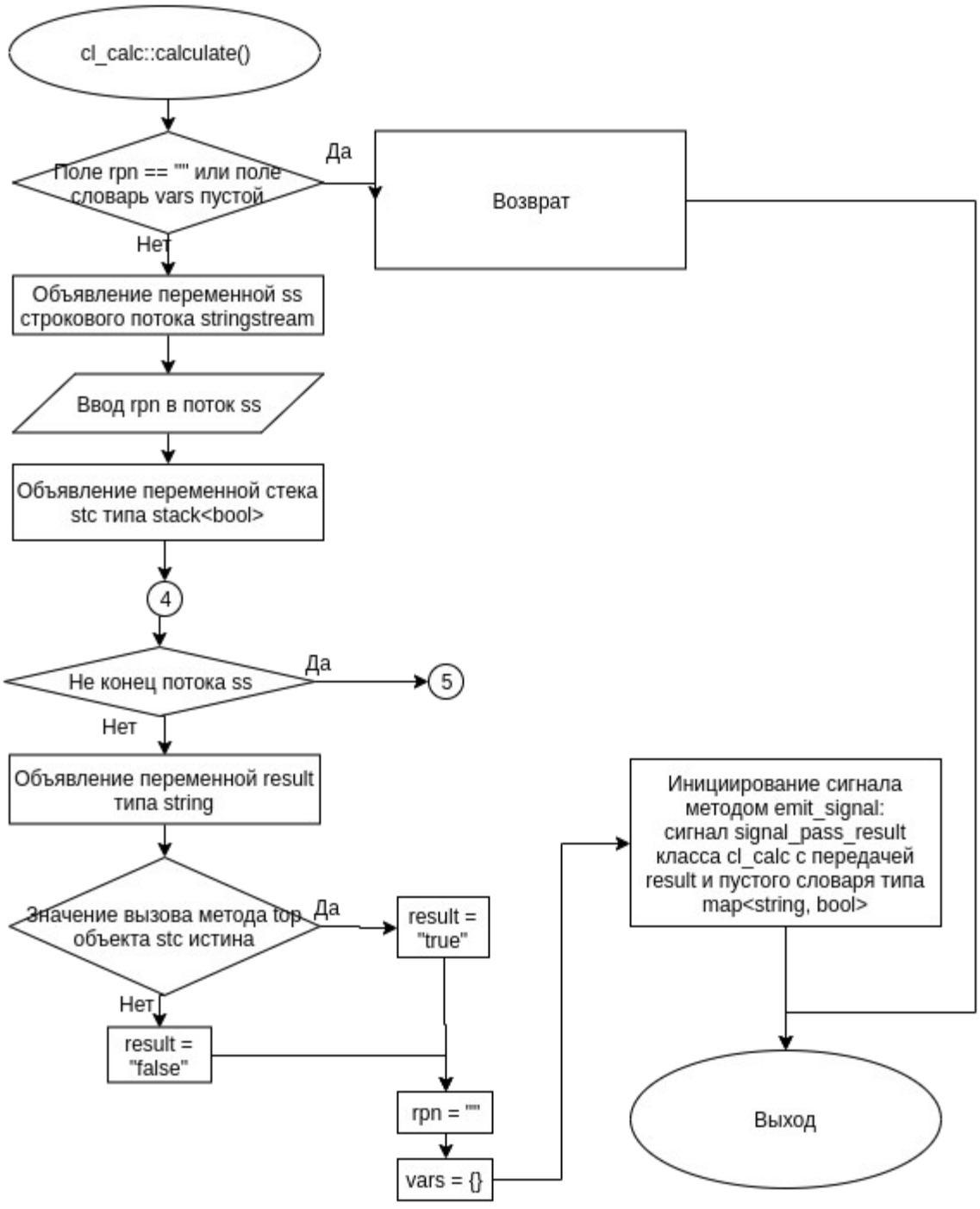

Рисунок 9 – Блок-схема алгоритма

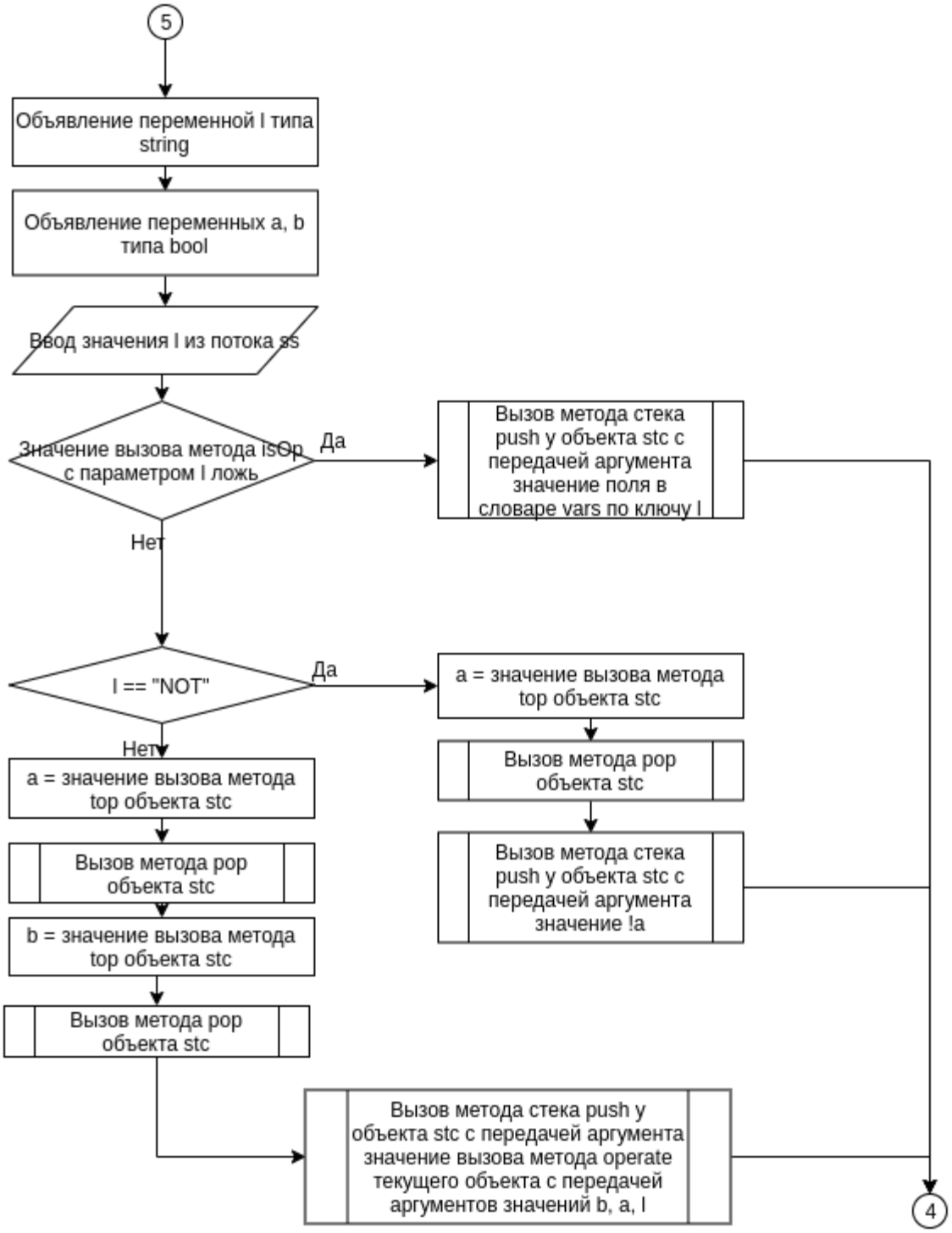

Рисунок 10 – Блок-схема алгоритма

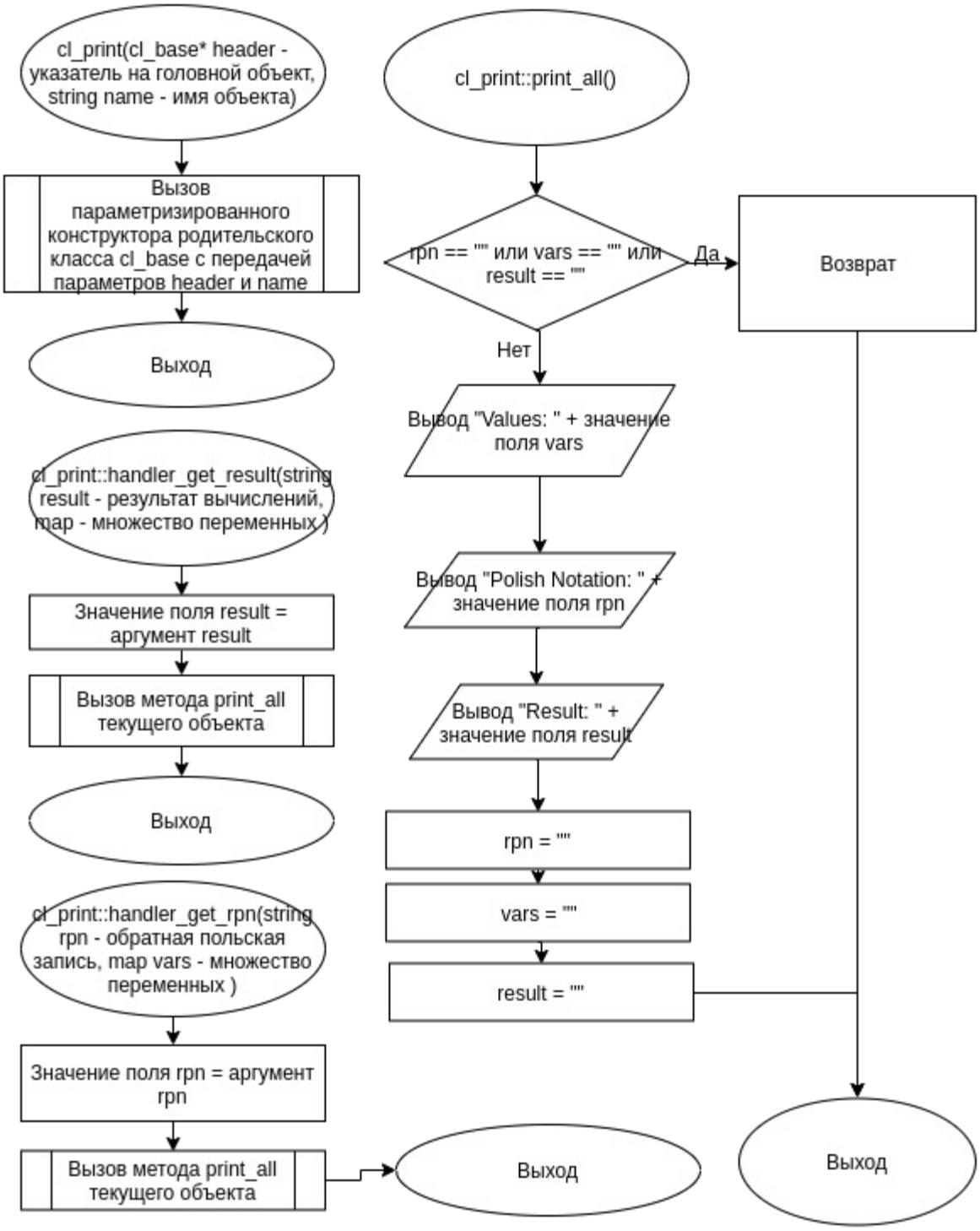

Рисунок 11 – Блок-схема алгоритма

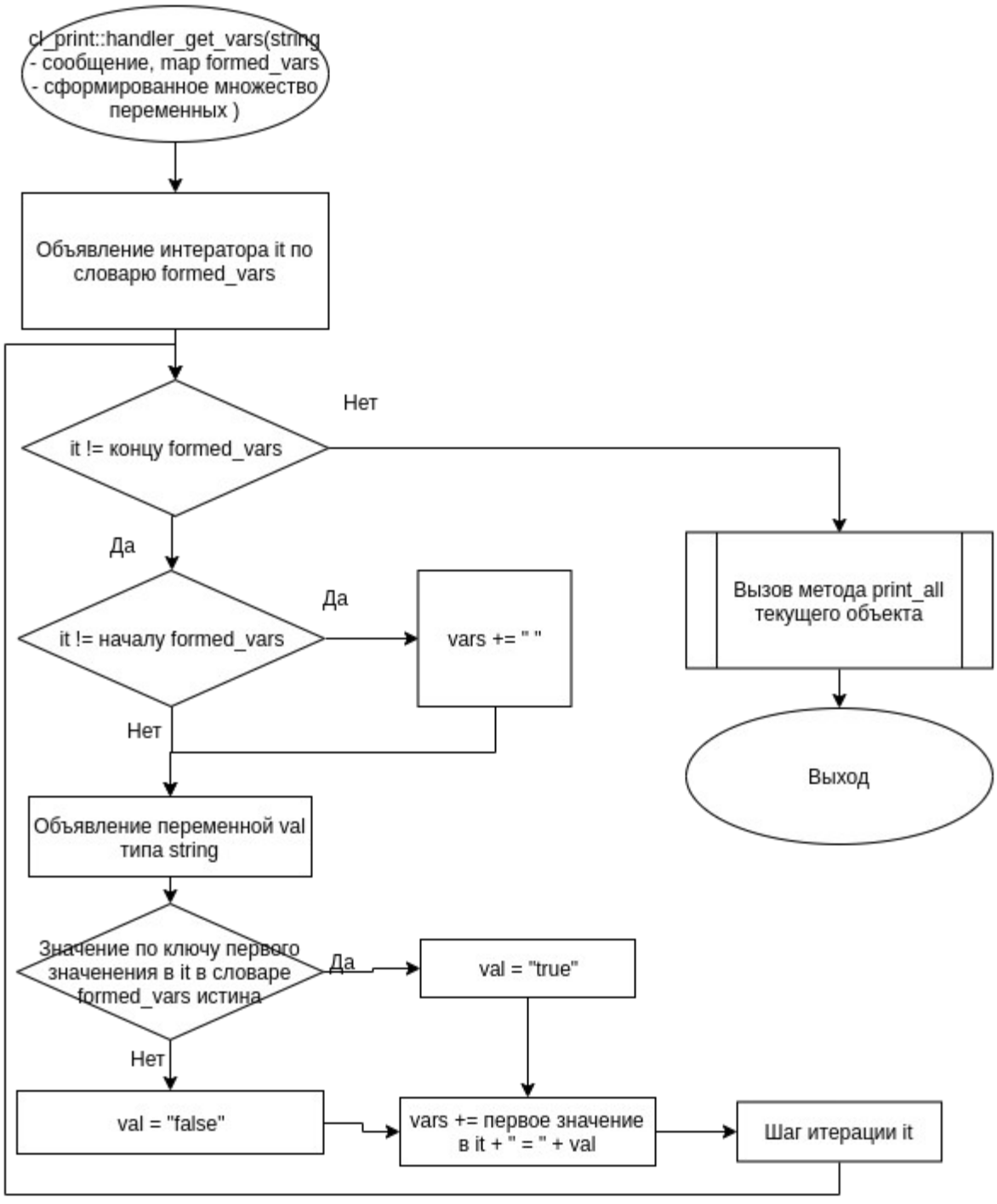

Рисунок 12 – Блок-схема алгоритма

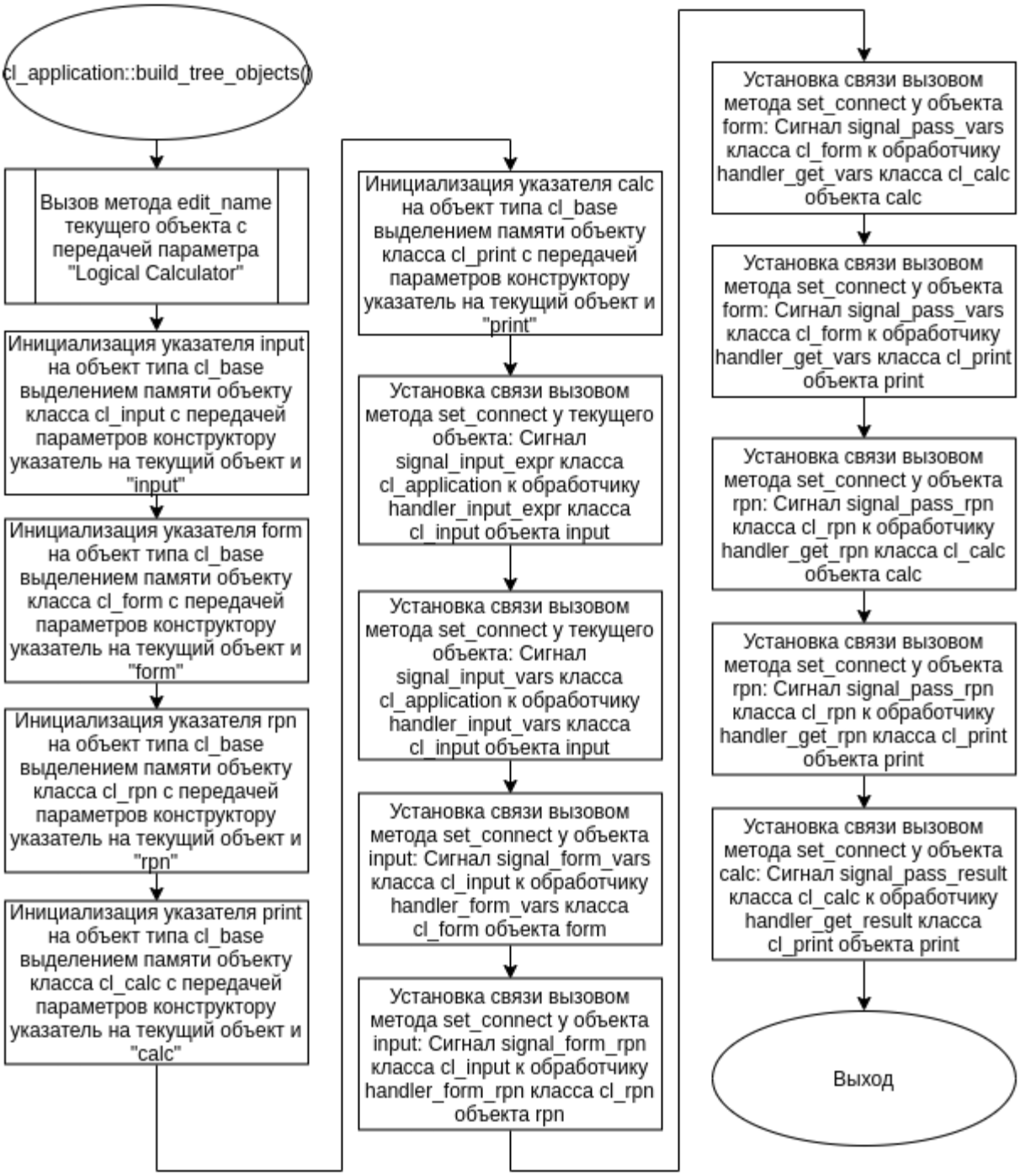

Рисунок 13 – Блок-схема алгоритма

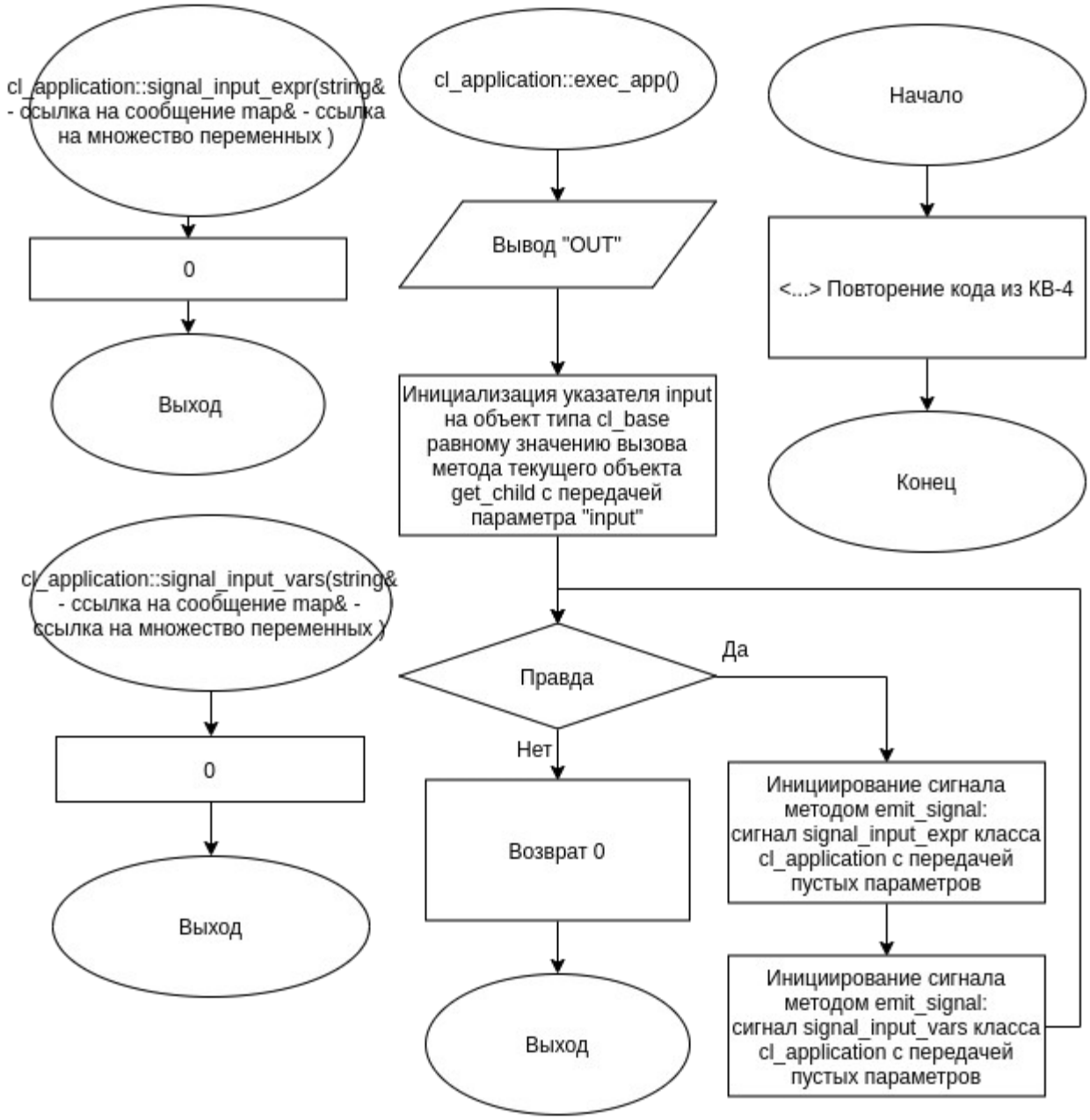

Рисунок 14 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_application.cpp

Листинг 1 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_input.h"
#include "cl_form.h"
#include "cl_rpn.h"
#include "cl_print.h"
#include "cl_calc.h"

cl_application::cl_application(cl_base* header) : cl_base(header) {};


void cl_application::signal_input_expr(string&, map<string, bool>&) {}
void cl_application::signal_input_vars(string&, map<string, bool>&) {}

void cl_application::build_tree_objects() {
   this->edit_name("Logical Calculator");

   cl_base* input = new cl_input(this, "input");
   cl_base* form = new cl_form(this, "form");
   cl_base* rpn = new cl_rpn(this, "rpn");
   cl_base* calc = new cl_calc(this, "calc");
   cl_base* print = new cl_print(this, "print");

   set_connect(SIGNAL_D(cl_application::signal_input_expr),            input,
HANDLER_D(cl_input::handler_input_expr));
   set_connect(SIGNAL_D(cl_application::signal_input_vars),            input,
HANDLER_D(cl_input::handler_input_vars));

   input->set_connect(SIGNAL_D(cl_input::signal_form_vars),             form,
HANDLER_D(cl_form::handler_form_vars));
   input->set_connect(SIGNAL_D(cl_input::signal_form_rpn),               rpn,
HANDLER_D(cl_rpn::handler_form_rpn));

   form->set_connect(SIGNAL_D(cl_form::signal_pass_vars),               calc,
HANDLER_D(cl_calc::handler_get_vars));
   form->set_connect(SIGNAL_D(cl_form::signal_pass_vars),             print,
HANDLER_D(cl_print::handler_get_vars));

   rpn->set_connect(SIGNAL_D(cl_rpn::signal_pass_rpn),                  calc,
HANDLER_D(cl_calc::handler_get_rpn));
   rpn->set_connect(SIGNAL_D(cl_rpn::signal_pass_rpn),                print,
```

```
HANDLER_D(cl_print::handler_get_rpn));

   calc->set_connect(SIGNAL_D(cl_calc::signal_pass_result),            print,
HANDLER_D(cl_print::handler_get_result));
}


int cl_application::exec_app()
{
   cout << "OUT";

   cl_base* input = get_child("input");

   string __;
   map<string, bool> _;
   while(true) {
      emit_signal(SIGNAL_D(cl_application::signal_input_expr), __, _);
      emit_signal(SIGNAL_D(cl_application::signal_input_vars), __, _);
   }
   return 0;
};
```

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"


class cl_application : public cl_base {
public:
   cl_application(cl_base* header);
   void build_tree_objects();
   int exec_app();

   void signal_input_expr(string&, map<string, bool>&);
   void signal_input_vars(string&, map<string, bool>&);
};

#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"


cl_base::cl_base(cl_base* header, string name){
   this->name = name;
   this->header_ptr = header;
   if(header_ptr){
      header_ptr-> children_ptr.push_back(this);
   }
}

bool cl_base::edit_name(string new_name){
   this->name = new_name;
   return true;
}

string cl_base::get_name(){
   return this->name;
}

cl_base* cl_base::get_header(){
   return this->header_ptr;
}

cl_base* cl_base::get_child(string name){
   for(cl_base* child : children_ptr) {
      if(child->name == name) return child;
   }
   return nullptr;
}

cl_base* cl_base::find_on_branch(string name, int* count) {
   int c = 0;
   if(count == nullptr){
      count = &c;
   }

   cl_base* fchild = nullptr;
   if(this->name == name){
      fchild = this;
      (*count)++;
   }

   for(cl_base* child : children_ptr) {
      cl_base* f = child->find_on_branch(name, count);
      if(f)
         fchild = f;
   }

   if( (*count)==1 && fchild) {
```

```
            return fchild;
        }
        return nullptr;
    }

    cl_base* cl_base::find_on_tree(string name) {
        cl_base* fheader = this;
        while(fheader->get_header()){
            fheader = fheader->get_header();
        }
        return fheader->find_on_branch(name);
    }

    // КВ-3
    bool cl_base::change_header(cl_base* new_header) {
        // Нельзя переопределять корневой и создавать новый корень
        if(header_ptr == nullptr || new_header == nullptr)
            return false;

        // У нового головного нельзя чтобы появились два подчиненных объекта с
    одинаковым наименованием.
        if(new_header->get_child(name))
            return false;


        // Новый объект Не должен принадлежать к ветке текущего
        cl_base* current = new_header;
        while(current->get_header()){
            if(current == this) return false;
            current = current->get_header();
        }


        // Удаляем у текущего родителя текущий объект из списка подчинённых
        this->header_ptr->children_ptr.erase(find(this->header_ptr-
    >children_ptr.begin(), this->header_ptr->children_ptr.end(), this));

        // Переопределяем головной объект
        this->header_ptr = new_header;
        new_header->children_ptr.push_back(this);

        return true;
    }

    void cl_base::delete_by_name(string name) {
        for(int i = 0; i < children_ptr.size(); i++) {
            if(children_ptr[i]->name == name){
                children_ptr.erase(children_ptr.begin() + i);
                return;
            }
        }
    }

    cl_base* cl_base::find_by_coord(string coord) {
        string s_name;
```

```
// Пустая координата
if(coord == "")
   return nullptr;

cl_base* root = this;
while(root->get_header()){
   root = root->get_header();
}

// Только Корень
if(coord == "/") {
   return root;
}

// Поиск по уникальной имени от корневого
if(coord.substr(0,2)=="//") {
   s_name = coord.substr(2);
   return this->find_on_tree(s_name);
}

// Только текущий объект
if(coord == ".")
   return this;

// Поиск по уникальной имени от текущего
if(coord.substr(0,1)==".") {
   s_name = coord.substr(1);
   return this->find_on_branch(s_name);
}

// Если в начале стоит / преобразуем абсолютную координату
// в относительную и ищем относительную от корня
cl_base* current = this;

// Абсолютная координата от корневого объекта
if(coord.substr(0,1) == "/")
{
   current = root;
   coord = coord.substr(1);
}


// Относительная координата от текущего объекта
while(coord.find("/") != string::npos)  {
   int nsl = coord.find("/", 1);
   current = current->get_child(coord.substr(0, nsl));

   if(!current) {
      return nullptr;
   }

   coord = coord.substr(nsl + 1);
}
```

```
   return current->get_child(coord);
}

// КВ-3


// КВ-4
string cl_base::get_path()
{
   string path = "";
   cl_base* p_obj = this;
   while(p_obj->get_header()){
      path = "/" + p_obj->get_name() + path;
      p_obj = p_obj->get_header();
   }
   if(path == "") path = "/";
   return path;
}


void  cl_base  ::  set_connect  (TYPE_SIGNAL  p_signal,  cl_base*  p_object,
TYPE_HANDLER p_ob_handler)
{
   o_sh  * p_value;
   //-------------------------------------------------------
   // Цикл для исключения повторного установления связи
   for (unsigned int i = 0; i < connects.size (); i++)
   {
      if ( connects [ i ] -> p_signal  == p_signal     &&
           connects [ i ] -> p_cl_base == p_object     &&
           connects [ i ] -> p_handler == p_ob_handler    )
      {
         return;
      }
   }

   p_value =  new o_sh ( );          // создание объекта структуры для
                                    // хранения информации о новой связи
   p_value->p_signal = p_signal;
   p_value->p_cl_base = p_object;
   p_value->p_handler = p_ob_handler;

   connects.push_back ( p_value );         // добавление новой связи
}

void  cl_base::remove_connect  (TYPE_SIGNAL  p_signal,  cl_base*  p_object,
TYPE_HANDLER p_ob_handler) {
   for (auto i = connects.begin(); i < connects.end(); i++){
      o_sh* c = *i;
      if (c->p_signal == p_signal &&
         c->p_cl_base == p_object &&
         c->p_handler == p_ob_handler)
      {
         connects.erase(i);
         delete c;
```

```
            return;
         }
      }
   }

   // Добавлен параметр
   void cl_base :: emit_signal ( TYPE_SIGNAL p_signal, string & s_command,
   map<string, bool> & p_vars )
   {
      TYPE_HANDLER   p_handler;
      cl_base      * p_object;
      //-------------------------------------------------------------------------
   -
      if(this->state == 0) return;
      (this->*p_signal) (s_command, p_vars);  // вызов метода сигнала
      for ( unsigned int i = 0; i < connects.size ( ); i ++ ) // цикл по всем
   обработчикам
      {
         if ( connects [ i ] -> p_signal == p_signal )         // определение
   допустимого обработчика
         {
            p_handler = connects [ i ] -> p_handler;
            p_object  = connects [ i ] -> p_cl_base;
            if(p_object->state != 0)
               (p_object ->* p_handler ) ( s_command, p_vars );       // вызов
   метода обработчика
         }
      }
   }

   // КВ-4



   void cl_base::print() {
      cout << this->name;

      int tab = 0;
      cl_base* par = this;
      while(par->get_header()){
         par = par->get_header();
         tab += 1;
      }


      if(!children_ptr.empty())
      {
         for(cl_base* child : children_ptr) {
            cout << endl;
            for(int i = 0; i <= tab; i++) cout << "    ";
            child->print();
         }
      }
   }
```

```
void cl_base::print_state() {
   cout << this->name;
   if(this->state == 0) cout << " is not ready";
   else cout << " is ready";

   int tab = 0;
   cl_base* par = this;
   while(par->get_header()){
      par = par->get_header();
      tab += 1;
   }


   if(!children_ptr.empty())
   {
      for(cl_base* child : children_ptr) {
         cout << endl;
         for(int i = 0; i <= tab; i++) cout << "    ";
         child->print_state();
      }
   }
}

void cl_base::set_state(int state) {
   if(header_ptr && header_ptr->state == 0) this->state = 0;
   else this->state = state;

   if(state == 0) {
      for(cl_base* child : children_ptr){
         child->set_state(0);
      }
   }
}
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <string>
#include <vector>
#include <iostream>
#include <algorithm>
#include <map>
#include <stack>
#include <sstream>
```

```
using namespace std;



class cl_base {
protected:
   string name;
   cl_base *header_ptr;
   vector<cl_base*> children_ptr;
   int state = 1;

   #define SIGNAL_D( signal_f ) ( TYPE_SIGNAL ) ( & signal_f )
   #define HANDLER_D( handler_f ) ( TYPE_HANDLER ) ( & handler_f )

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


   struct o_sh                  // Структура задания одной связи
   {
      TYPE_SIGNAL  p_signal;    // Указатель на метод сигнала
      cl_base    * p_cl_base;   // Указатель на целевой объект
      TYPE_HANDLER p_handler;   // Указатель на метод обработчика
   };

   vector < o_sh * > connects;


public:
   cl_base(cl_base* base, string name = "Object");
   bool edit_name(string new_name);
   string get_name();
   cl_base* get_header();
   cl_base* get_child(string name);

   cl_base* find_on_branch(string name, int* count = nullptr); // Поиск на
ветке
   cl_base* find_on_tree(string name); // Поиск на всем дереве

   // КВ-3
   bool  change_header(cl_base*  new_header);  //  Переопределение  головного
объекта
   void delete_by_name(string name); // Удаление объекта по наименованию
   cl_base* find_by_coord(string coord); // Поиск по координате
   // КВ-3

   // КВ-4

   string get_path();
   void  set_connect  (TYPE_SIGNAL  p_signal,  cl_base*  p_object,  TYPE_HANDLER
p_ob_handler);
   void remove_connect (TYPE_SIGNAL p_signal, cl_base* p_object, TYPE_HANDLER
p_ob_handler);
   void  emit_signal  (TYPE_SIGNAL  p_signal,  string&  s_command,  map<string,
```

```
bool> &);
   // KB-4


   void print();
   void print_state();

   void set_state(int state);

};




#endif
```

5.5 Файл cl_calc.cpp

Листинг 5 – cl_calc.cpp

```
#include "cl_calc.h"

cl_calc::cl_calc(cl_base* header, string name) : cl_base(header, name) {}

void cl_calc::signal_pass_result(string&, map<string, bool>&) {}

void cl_calc::handler_get_rpn(string rpn, map<string, bool>){
   this->rpn = rpn;
   calculate();
}

void cl_calc::handler_get_vars(string, map<string, bool> vars){
   this->vars = vars;
   calculate();
}

void cl_calc::calculate() {
   if(rpn == "" || vars.empty()) return;

   stringstream ss;
   ss << rpn;
   stack<bool> stc;

   while(!ss.eof()) {
      string l;
      bool a, b;
      ss >> l;
      if(!isOp(l)) {
         stc.push(vars[l]);
      }
```

```
            else if(l == "NOT") {
               a = stc.top();
               stc.pop();
               stc.push(!a);
            }
            else {
               a = stc.top();
               stc.pop();
               b = stc.top();
               stc.pop();
               stc.push(operate(b, a, l));
            }
         }

         string result;
         if(stc.top()) result = "true";
         else result = "false";

         rpn = "";
         vars = {};

         map<string, bool> _;
         emit_signal(SIGNAL_D(cl_calc::signal_pass_result), result, _);

      }

      bool cl_calc::operate(bool a, bool b, string op) {
         if(op == "AND") return a && b;
         else if(op == "XOR") return a != b;
         else if(op == "OR") return a || b;
         else if(op == "=>") return !a || b;
         else if(op == "<=>") return a == b;
         else return a;
      }

      bool cl_calc::isOp(string op) {
         return op == "NOT" || op == "AND" || op == "XOR" || op == "OR" || op ==
      "<=>" || op == "=>";
      }
```

5.6 Файл cl_calc.h

Листинг 6 – cl_calc.h

```
      #ifndef __CL_CALC__H
      #define __CL_CALC__H
      #include "cl_base.h"

      class cl_calc : public cl_base
```

```
{
private:
   string rpn = "";
   map<string, bool> vars;
public:
   cl_calc(cl_base* header, string name);

   void signal_pass_result(string&, map<string, bool>&);
   void handler_get_rpn(string, map<string, bool>);
   void handler_get_vars(string, map<string, bool>);
private:
   void calculate();
   bool operate(bool a, bool b, string op);
   bool isOp(string op);
};



#endif
```

5.7 Файл cl_form.cpp

Листинг 7 – cl_form.cpp

```
#include "cl_form.h"

cl_form::cl_form(cl_base* header, string name) : cl_base(header, name) {}

void cl_form::signal_pass_vars(string&, map<string, bool>&) {}

void cl_form::handler_form_vars(string vars, map<string, bool>) {
   stringstream ss;
   ss << vars;
   map<string, bool> formed_vars;

   while(!ss.eof()) {
      string symbol, eql, val;
      ss >> symbol >> eql >> val;

      bool v;
      if(val == "1") v = true;
      else v = false;
      formed_vars.insert({symbol, v});
   }

   string _ = "";
   emit_signal(SIGNAL_D(cl_form::signal_pass_vars), _, formed_vars);
}
```

5.8 Файл cl_form.h

Листинг 8 – cl_form.h

```
#ifndef __CL_FORM__H
#define __CL_FORM__H
#include "cl_base.h"


class cl_form : public cl_base
{
public:
   cl_form(cl_base* header, string name);
   void signal_pass_vars(string&, map<string, bool>&);
   void handler_form_vars(string, map<string, bool>);
};

#endif
```

5.9 Файл cl_input.cpp

Листинг 9 – cl_input.cpp

```
#include "cl_input.h"
#include "cl_form.h"


cl_input::cl_input(cl_base* header, string name) : cl_base(header, name) {}

void cl_input::signal_form_rpn(string&, map<string, bool>&) {}
void cl_input::signal_form_vars(string&, map<string, bool>&) {}

void cl_input::handler_input_expr(string, map<string, bool>) {
   string expression;
   getline(cin, expression);

   if(expression == ".") {
      exit(0);
   }
   if(expression == "SHOWTREE") {
      cout << endl;
      header_ptr->print_state();
      exit(0);
   }

   map<string, bool> _;
   emit_signal(SIGNAL_D(cl_input::signal_form_rpn), expression, _);
}
```

```
void cl_input::handler_input_vars(string, map<string, bool>) {
   //cout << endl << "INPUT ВВЁЛ ПЕРЕМЕННЫЕ ";

   string vars;
   getline(cin, vars);

   map<string, bool> _;
   emit_signal(SIGNAL_D(cl_input::signal_form_vars), vars, _);
}
```

5.10 Файл cl_input.h

Листинг 10 – cl_input.h

```
#ifndef __CL_INPUT__H
#define __CL_INPUT__H
#include "cl_base.h"

class cl_input : public cl_base
{
public:
   cl_input(cl_base* header, string name);
   void signal_form_vars(string&, map<string, bool>&);
   void signal_form_rpn(string&, map<string, bool>&);
   void handler_input_expr(string, map<string, bool>);
   void handler_input_vars(string, map<string, bool>);
};

#endif
```

5.11 Файл cl_print.cpp

Листинг 11 – cl_print.cpp

```
#include "cl_print.h"

cl_print::cl_print(cl_base* header, string name) : cl_base(header, name) {}


void cl_print::handler_get_rpn(string rpn, map<string, bool>) {
   this->rpn = rpn;
   print_all();
}
```

```
void cl_print::handler_get_vars(string, map<string, bool> formed_vars) {
   for(auto it = formed_vars.begin(); it != formed_vars.end(); it++){
      if(it != formed_vars.begin()) vars += " ";
      string val;
      if(formed_vars[it->first]) val = "true";
      else val = "false";
      vars += it->first + " = " + val;
   }

   print_all();
}

void cl_print::handler_get_result(string result, map<string, bool>) {
   this->result = result;

   print_all();
}

void cl_print::print_all() {
   if(rpn == "" || vars == "" || result == "") return;

   cout << endl << "Values: " << vars;
   cout << endl << "Polish Notation: " << rpn;
   cout << endl << "Result: " << result;

   rpn = "";
   vars = "";
   result = "";
}
```

5.12 Файл cl_print.h

Листинг 12 – cl_print.h

```
#ifndef __CL_PRINT__H
#define __CL_PRINT__H
#include "cl_base.h"

class cl_print : public cl_base
{
private:
   string rpn = "";
   string vars = "";
   string result = "";
public:
   cl_print(cl_base* header, string name);

   void handler_get_vars(string, map<string, bool>);
   void handler_get_rpn(string, map<string, bool>);
   void handler_get_result(string, map<string, bool>);
```

```
   void print_all();
};


#endif
```

5.13 Файл cl_rpn.cpp

Листинг 13 – cl_rpn.cpp

```
#include "cl_rpn.h"

cl_rpn::cl_rpn(cl_base* header, string name) : cl_base(header, name) {}


void cl_rpn::signal_pass_rpn(string&, map<string, bool>&) {}


void cl_rpn::handler_form_rpn(string expression, map<string, bool>) {
   map <string, int> priority_map = {{"NOT", 5}, {"AND", 4}, {"XOR", 3},
{"OR", 2}, {"=>", 1}, {"<=>", 0}};
   stringstream ss;
   ss << expression;
   stack<string> stc;
   string result = "";

   while(!ss.eof()) {
      string l;
      ss >> l;
      if(l.find('.') != -1)
         l.pop_back();

      if(!isOp(l) && l != "(" && l != ")")
         result += l + " ";

      else if (l == "NOT" || l == "(")
         stc.push(l);

      else if (l == ")"){
         while(stc.top() != "(") {
            result += stc.top() + " ";
            stc.pop();
         }
         stc.pop();
      }

      else if (isOp(l)) {
         if(!stc.empty()) {
            while(stc.top()   ==   "NOT"   ||   (priority_map[stc.top()]   >=
```

```
priority_map[l])){
                result += stc.top() + " ";
                stc.pop();
                if(stc.empty())
                   break;
             }
          }
          stc.push(l);
       }
    }
    while(!stc.empty()) {
       result += stc.top() + " ";
       stc.pop();
    }
    result.pop_back();

    map<string, bool> _ = {};
    emit_signal(SIGNAL_D(cl_rpn::signal_pass_rpn), result, _);
}

bool cl_rpn::isOp(string op) {
    return op == "NOT" || op == "AND" || op == "XOR" || op == "OR" || op ==
"<=>" || op == "=>";
}
```

5.14 Файл cl_rpn.h

Листинг 14 – cl_rpn.h

```
#ifndef __CL_RPN__H
#define __CL_RPN__H
#include "cl_base.h"

class cl_rpn : public cl_base
{
public:
    cl_rpn(cl_base* header, string name);
    void signal_pass_rpn(string&, map<string, bool>&);
    void handler_form_rpn(string, map<string, bool>);
private:
    bool isOp(string op);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>

#include "cl_application.h"


int main()
{
  cl_application   ob_cl_application  (  nullptr  );  //  создание  корневого
объекта
  ob_cl_application.build_tree_objects  (  );            //  конструирование
системы, построение дерева объектов
  return ob_cl_application.exec_app ( );     // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 31.

Таблица 31 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
.	OUT	OUT
SHOWTREE	OUT Logical Calculator is ready input is ready form is ready rpn is ready calc is ready print is ready	OUT Logical Calculator is ready input is ready form is ready rpn is ready calc is ready print is ready
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
c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 0 b = 0 c = 0 c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 0 b = 0 c = 1 c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 0 b = 1 c = 0 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 c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 1 b = 0 c = 1 c <=> NOT (a XOR b	OUT Values: a = false b = false c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = false b = false c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = false b = true c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: false Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=>	OUT Values: a = false b = false c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = false b = false c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = false b = true c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: false Values: a = false b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=>

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
OR a AND c) => b XOR c. a = 1 b = 1 c = 0 c <=> NOT (a XOR b OR a AND c) => b XOR c. a = 1 b = 1 c = 1 .	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 Values: a = true b = false c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = true b = true c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: false Values: a = true b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true	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 Values: a = true b = false c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true Values: a = true b = true c = false Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: false Values: a = true b = true c = true Polish Notation: c a b XOR a c AND OR NOT b c XOR => <=> Result: true
a XOR b OR (NOT a => b) <=> c. a = 0 b = 1 c = 1 a XOR b OR (NOT a => b) <=> c. a = 1 b = 1 c = 0 SHOWTREE	OUT Values: a = false b = true c = true Polish Notation: a b XOR a NOT b => OR c <=> Result: true Values: a = true b = true c = false Polish Notation: a b XOR a NOT b => OR c <=> Result: false Logical Calculator is ready input is ready form is ready rpn is ready calc is ready print is ready	OUT Values: a = false b = true c = true Polish Notation: a b XOR a NOT b => OR c <=> Result: true Values: a = true b = true c = false Polish Notation: a b XOR a NOT b => OR c <=> Result: false Logical Calculator is ready input is ready form is ready rpn is ready calc is ready print is ready

ЗАКЛЮЧЕНИЕ

При выполнении работы были достигнуты все поставленные задачи:

- Освоение принципов объектно-ориентированного программирования;
- Освоение основ объектно-ориентированного программирования;
- Освоение умения проектирования архитектуры программы на базе построения иерархии объектов;
- Освоение выполнения всех необходимых работ согласно этапам разработки программы и соответствующих программных инструментов;
- Моделироание работы логического калькулятора при помощи сигналов и обработчиков;
- Построение системы взаимодействия объектов с помощью интерфейса сигналов и обработчиков;
- Описание алгоритма работы программы;
- Построение блок-схем для разработанного алгоритма;
- Написание кода на языке программирования C++, согласно разработанному алгоритму работы программы;
- Тестирование работы программы.

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).