Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов. В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Каждый объект на дереве иерархии имеет свое место и наименование. Не допускается для одного головного объекта одинаковые наименования в составе подчиненных объектов.

Создать базовый класс со следующими элементами:

- Свойства:
  - Наименование объекта (строкового типа);
  - Указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно nullptr);
  - Динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.
- Функционал:
  - Параметризированный конструктор с параметрами: указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии; строкового типа, содержащий наименование создаваемого объекта (имеет значение по умолчанию);
  - Метод редактирования имени объекта. Один параметр строкового типа, содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
  - Метод получения имени объекта;

- Метод получения указателя на головной объект текущего объекта;
- Метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- Метод получения указателя на непосредственно подчиненный объект по его имени. Если объект не найден, то возвращает nullptr. Один параметр строкового типа, содержит наименование искомого подчиненного объекта.

Для построения дерева иерархии объектов в качестве корневого объекта используется объект приложение. Класс объекта приложения наследуется от базового класса. Объект приложение реализует следующий функционал:

- Метод построения исходного дерева иерархии объектов (конструирования моделируемой системы);
- Метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз. Переход на новый уровень происходит только от правого (последнего) объекта предыдущего уровня. Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Исключить создание объекта если его наименование совпадает с именем уже имеющегося подчиненного объекта у предполагаемого головного. Исключить добавление нового объекта, не последнему подчиненному предыдущего уровня.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
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
```

```
}
```

Наименование класса cl_application и идентификатора корневого объекта ob_cl_application могут быть изменены разработчиком.

Все версии курсовой работы имеют такую основную функцию.

1.1 Описание входных данных

Первая строка:

```
«имя корневого объекта»
```

Вторая строка и последующие строки:

```
«имя головного объекта» «имя подчиненного объекта»
```

Создается подчиненный объект и добавляется в иерархическое дерево. Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

Пример ввода

```
Object_root
Object_root Object_1
Object_root Object_2
Object_root Object_3
Object_3 Object_4
Object_3 Object_5
Object_6 Object_6
```

Дерево объектов, которое будет построено по данному примеру:

```
Object_root
    Object_1
    Object_2
    Object_3
        Object_4
        Object_5
```

1.2 Описание выходных данных

Первая строка:

«имя корневого объекта»

Вторая строка и последующие строки имена головного и подчиненных объектов очередного уровня разделенных двумя пробелами.

«имя головного объекта» «имя подчиненного объекта»[[«имя подчиненного объекта»] …….]

Пример вывода:

```
Object_root
Object_root  Object_1  Object_2  Object_3
Object_3  Object_4  Object_5
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используются:

- объект стандартного потока ввода std::cin (используется для ввода с клавиатуры);
- объект стандартного потока вывода std::cout (используется для вывода на экран);
- оператор цикла со счётчиком for;
- оператор цикла с условием while;
- условный оператор if..else;
- объекты классов cl_base, cl_node и cl_application.

Иерархия классов представлена в таблице 1.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	cl_base			Базовый класс, на основании которого создаются классы, из которых состоит дерево иерархии		

		cl_application	public		2	
		cl_node	public		3	
2	cl_application			Класс корневого объекта (объекта-приложения)		
3	cl_node			Класс объектов, из которых состоит дерево		

1. Класс cl_base:

- Свойства (поля):
  - Поле, отвечающее за хранение имени объекта:
    - Наименование - name;
    - Тип - строка;
    - Модификатор доступа - закрытый;
  - Поле, отвечающее за хранение указателя на головной объект в дереве иерархии:
    - Наименование - parent;
    - Тип - указатель на объект класса cl_base;
    - Модификатор доступа - закрытый;

    - Поле, отвечающее на хранение указателей на подчинённые объекты в дереве иерархии:
        - Наименование - children;
        - Тип - вектор указателей на объекты класса cl_base;
        - Модификатор доступа - закрытый;
- Методы:
    - Параметризованный конструктор:
        - Функционал - создание объекта класса cl_base, присвоение ему заданного имени и интеграция его в дереве иерархии;
        - Возвращаемое значение - объект класса cl_base;
        - Модификатор доступа - открытый;
        - Параметры:
            - parent - указатель на головной объект;
            - name - имя объекта;
    - Метод set_name:
        - Функционал - установка имени объекта класса cl_base;
        - Возвращаемое значение - логическое значение;
        - Модификатор доступа - открытый;
        - Параметры:
            - new_name - новое имя объекта;
    - Метод get_name:
        - Функционал - возвращает текущее имя объекта класса cl_base;
        - Возвращаемое значение - строковое значение;
        - Модификатор доступа - открытый;
        - Параметры - отсутствуют;
    - Метод get_parent:
        - Функционал - возвращает указатель на родительский объект;

        - Возвращаемое значение - указатель на объект класса cl_base;
        - Модификатор доступа - открытый;
        - Параметры - отсутствуют;
    - Метод print_tree:
        - Функционал - вывод наименований объектов в дереве иерархии сверху вниз и слева направо;
        - Возвращаемое значение - отсутствует;
        - Модификатор доступа - открытый;
        - Параметры - отсутствуют;
    - Метод get_child:
        - Функционал - поиск объекта с указанным именем среди объектов, подчинённых данному;
        - Возвращаемое значение - указатель на объект cl_base
        - Модификатор доступа - открытый;
        - Параметры:
            - child_name - имя искомого объекта;

2. Класс cl_application:

- Свойства (поля) - дополнительные свойства отсутствуют;
- Методы:
    - Параметризованный конструктор:
        - Функционал - создание объекта класса cl_application с заданным родительским объектом;
        - Возвращаемое значение - объект класса cl_application;
        - Модификатор доступа - открытый;
        - Параметры:
            - parent - указатель на родительский объект;
    - Метод build_tree_objects:

        - Функционал - создание исходного дерева иерархии объектов;
        - Возвращаемое значение - отсутствует;
        - Модификатор доступа - открытый;
        - Параметры - отсутствуют;
    - Метод exec_app:
        - Функционал - запуск приложения;
        - Возвращаемое значение - целочисленное значение;
        - Модификатор доступа - открытый;
        - Параметры - отсутствуют.

3. Класс cl_node:

- Свойства (поля) - дополнительные свойства отсутствуют;
- Методы:
    - Параметризованный конструктор:
        - Функционал - создание объекта класса cl_node с заданным именем и родительским объектом;
        - Возвращаемое значение - объект класса cl_node;
        - Модификатор доступа - открытый;
        - Параметры:
            - parent - указатель на родительский объект;
            - name - имя объекта.

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 функции main

Функционал: основной алгоритм программы.

Параметры: .

Возвращаемое значение: целочисленный код завершения работы программы.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_cl_application класса cl_application с использованием параметризованного конструктора и нулевого указателя (nullptr) в качестве параметра	2
2		Вызов метода build_tree_objects объекта ob_cl_application	3
3		Возвращения значения, возвращённого методом exec_app класса ob_cl_application	∅

3.2 Алгоритм конструктора класса cl_base

Функционал: создание объекта класса cl_base, присвоение ему заданного имени и интеграция его в дерево иерархии объектов.

Параметры: parent - указатель на головной объект, name - имя объекта.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Присваивание полю parent значения параметра parent	2
2		Присваивание полю name значения параметра name	3
3	Значение поля parent не равно нулевому указателю (nullptr)		4
			∅
4		Вызов метода push_back поля children объекта parent с указателем this в качестве аргумента	∅

3.3 Алгоритм метода set_name класса cl_base

Функционал: установка имени объекта класса cl_base.

Параметры: new_name - новое имя объекта.

Возвращаемое значение: логическое значение.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода
1	Значение поля parent не равно нулевому указателю (nullptr)		2
			6
2		Инициализация целочисленной переменной i значением 0	3
3	Значение переменной i меньше значения возвращённого методом size		4

№	Предикат	Действия	№ перехода
	поля children объекта parent		
			6
4	Значение поля name элемента поля children поля parent с индексом i равно значению параметра new_name	Возвращение значения логическая ложь (false)	∅
			5
5		Увеличение значения переменной i на 1	3
6		Присваивание полю name значения параметра new_name	7
7		Возвращение значения логическая истина (true)	∅

3.4 Алгоритм метода get_name класса cl_base

Функционал: возвращает текущее имя объекта класса cl_base.

Параметры: отсутствуют.

Возвращаемое значение: строковое значение.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращение значения поля name	∅

3.5 Алгоритм метода get_parent класса cl_base

Функционал: возвращает указатель на родительский объект класса cl_base.

Параметры: отсутствуют.

Возвращаемое значение: указатель на объект cl_base.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_parent класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращение значения поля parent	∅

3.6 Алгоритм метода print_tree класса cl_base

Функционал: вывод наименований объектов в дереве иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод значения поля name	2
2		Инициализация целочисленной переменной i значением 0	3
3	Значение переменной i меньше значения возвращённого методом size поля children	Вывод " " (2 пробела) и значения поля name элемента поля children с индексом i	4
			5
4		Увеличение значения переменной i на 1	3
5		Инициализация логической переменной last значением true	6
6		Инициализация целочисленной переменной i значеним 0	7
7	Значение переменной i меньше значения возвращённого методом size		8

№	Предикат	Действия	№ перехода
	поля children		
			∅
8	Значение возвращённое методом size поля children элемента поля children с индексом i не равно 0		9
			12
9	Значение переменной last равно логической истине	Вывод std::endl (переход на новую строку)	10
			11
10		Присваивание переменной last значения логической лжи (false)	11
11		Вызов метода print_tree элемента поля children с индексом i	12
12		Увеличение значения переменной i на 1	7

3.7 Алгоритм метода get_child класса cl_base

Функционал: поиск объекта с указанным именем среди объектов, подчинённых данному.

Параметры: child_name - имя искомого объекта.

Возвращаемое значение: указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get_child класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i значением 0	2

№	Предикат	Действия	№ перехода
2	Значение переменной i меньше значения возвращённого методом size поля children		3
			5
3	Значение поля name элемента поля children с индексом i равно значению параметра new_name	Возвращение элемента поля children с индексом i	∅
			4
4		Увеличение значения переменной i на 1	2
5		Возвращение нулевого указателя (nullptr)	∅

3.8 Алгоритм конструктора класса cl_node

Функционал: создание объекта класса cl_node с заданным именем и родительским объектом.

Параметры: parent - указатель на родительский объект, name - имя объекта.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_node

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса cl_base с параметрами parent и name	∅

3.9 Алгоритм конструктора класса cl_application

Функционал: создание объекта класса cl_application с заданным родительским объектом.

Параметры: parent - указатель на родительский объект.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса cl_base с параметром parent	∅

3.10 Алгоритм метода build_tree_objects класса cl_application

Функционал: создание исходного дерева иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной parent_name	2
2		Объявление строковой переменной child_name	3
3		Инициализация указателя на объект cl_base last_parent указателем this	4
4		Инициализация указателя на объект cl_base last_child указателем this	5
5		Ввод значения переменной parent_name	6
6		Вызов метода set_name с переменной parent_name в качестве параметра	7
7	Ввод значений переменных parent_name и child_name вернул значение истина		8
			∅

№	Предикат	Действия	№ перехода
8	Значение переменной parent_name равно значению переменной child_name		∅
			9
9	Значение переменной parent_name равно значению возвращённому методом get_name объекта last_parent и значение метода возвращённого методом get_child объекта last_child с переменной child_name в качестве аргумента равно нулевому указателю (nullptr)	Присваивание указателю last_child указатель на новый объект класса cl_node, созданный с помощью оператора new с использованием параметризованного конструктора и указателями last_parent и last_child в качестве параметров	7
			10
10	Значение переменной parent_name равно значению возвращённому методом get_name объекта last_child и значение метода возвращённого методом get_child объекта last_child с переменной child_name в качестве аргумента равно нулевому указателю (nullptr)	Присваивание переменной last_parent значения переменной last_child	11
			7
11		Присваивание указателю last_child указатель на новый объект класса cl_node, созданный с помощью оператора new с использованием параметризованного конструктора и указателями	7

№	Предикат	Действия	№ перехода
		last_parent и last_child в качестве параметров	

3.11 Алгоритм метода exec_app класса cl_application

Функционал: запуск приложения.

Параметры: отсутствуют.

Возвращаемое значение: целочисленный код завершения работы приложения.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод значения возвращённого методом get_name и std::endl (переход на новую строку)	2
2		Вызов метода print_tree	3
3		Возвращение целочисленного значения 0	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-18.

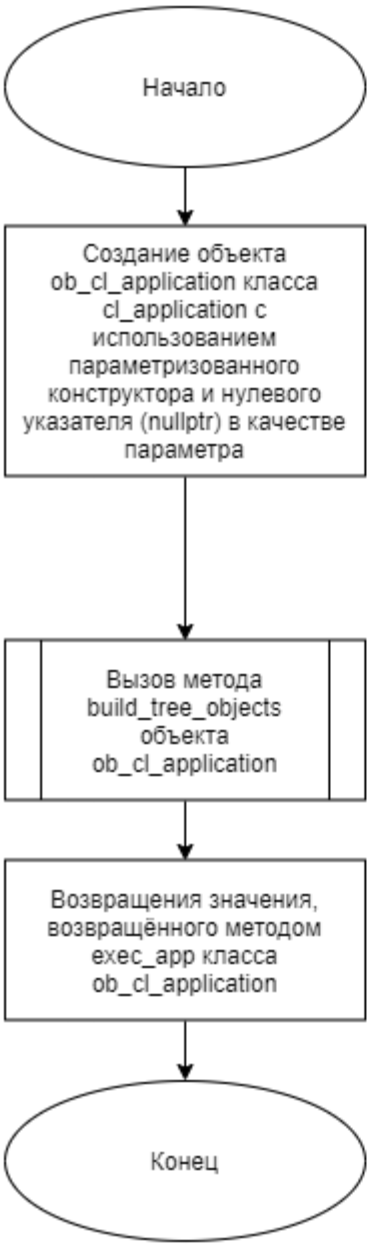

Рисунок 1 – Блок-схема алгоритма

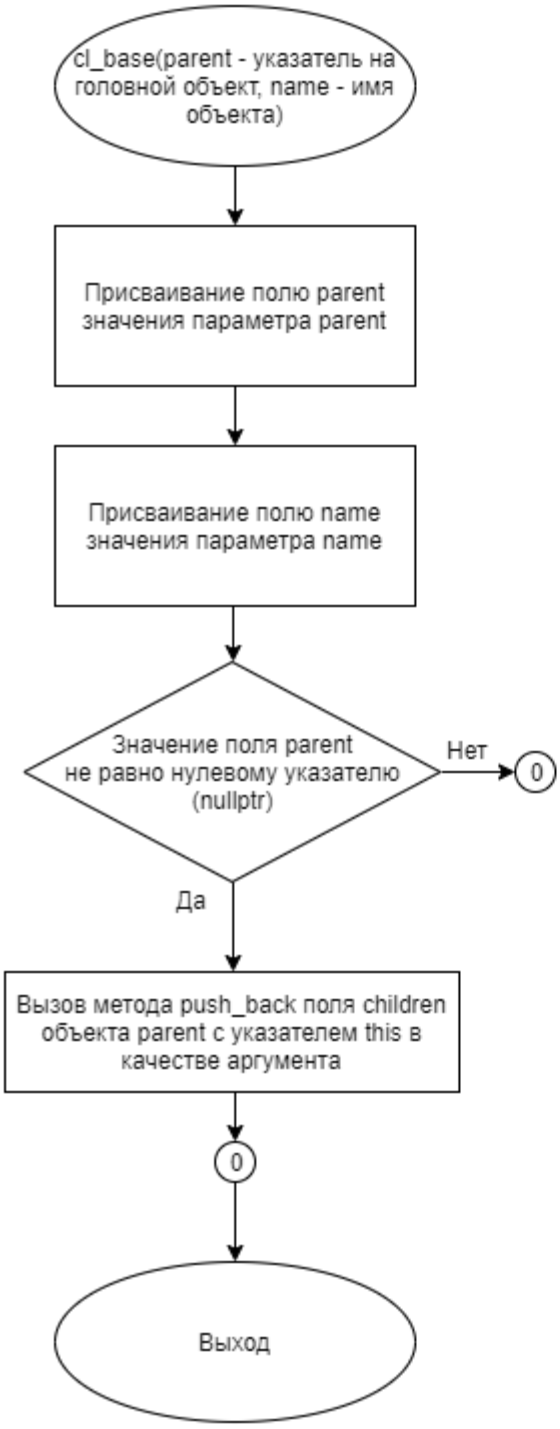

Рисунок 2 – Блок-схема алгоритма

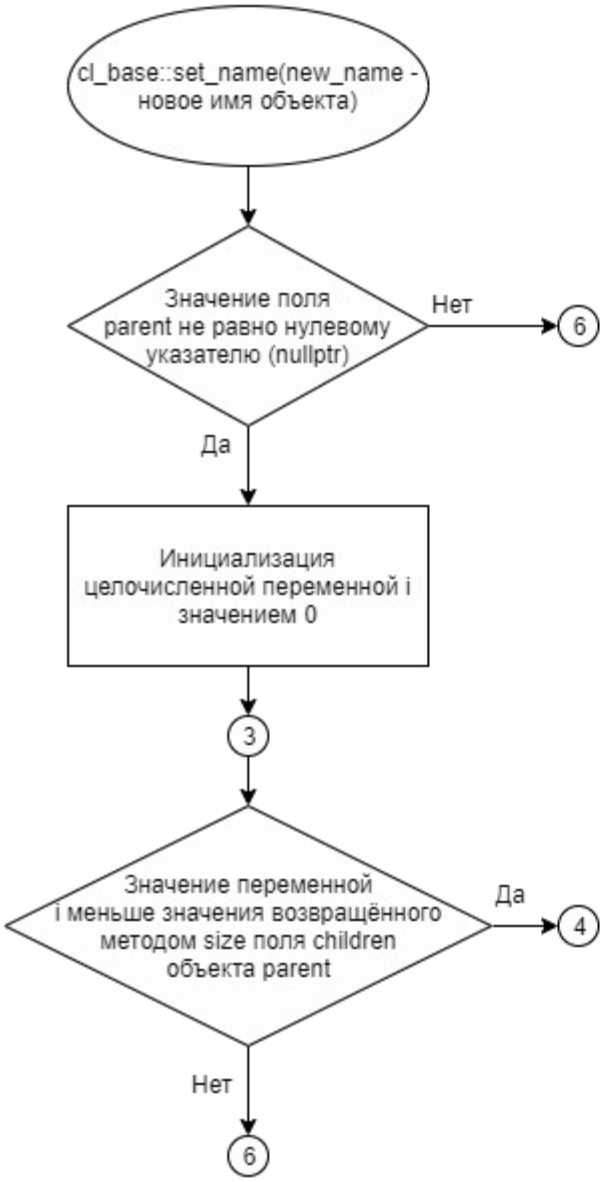

Рисунок 3 – Блок-схема алгоритма

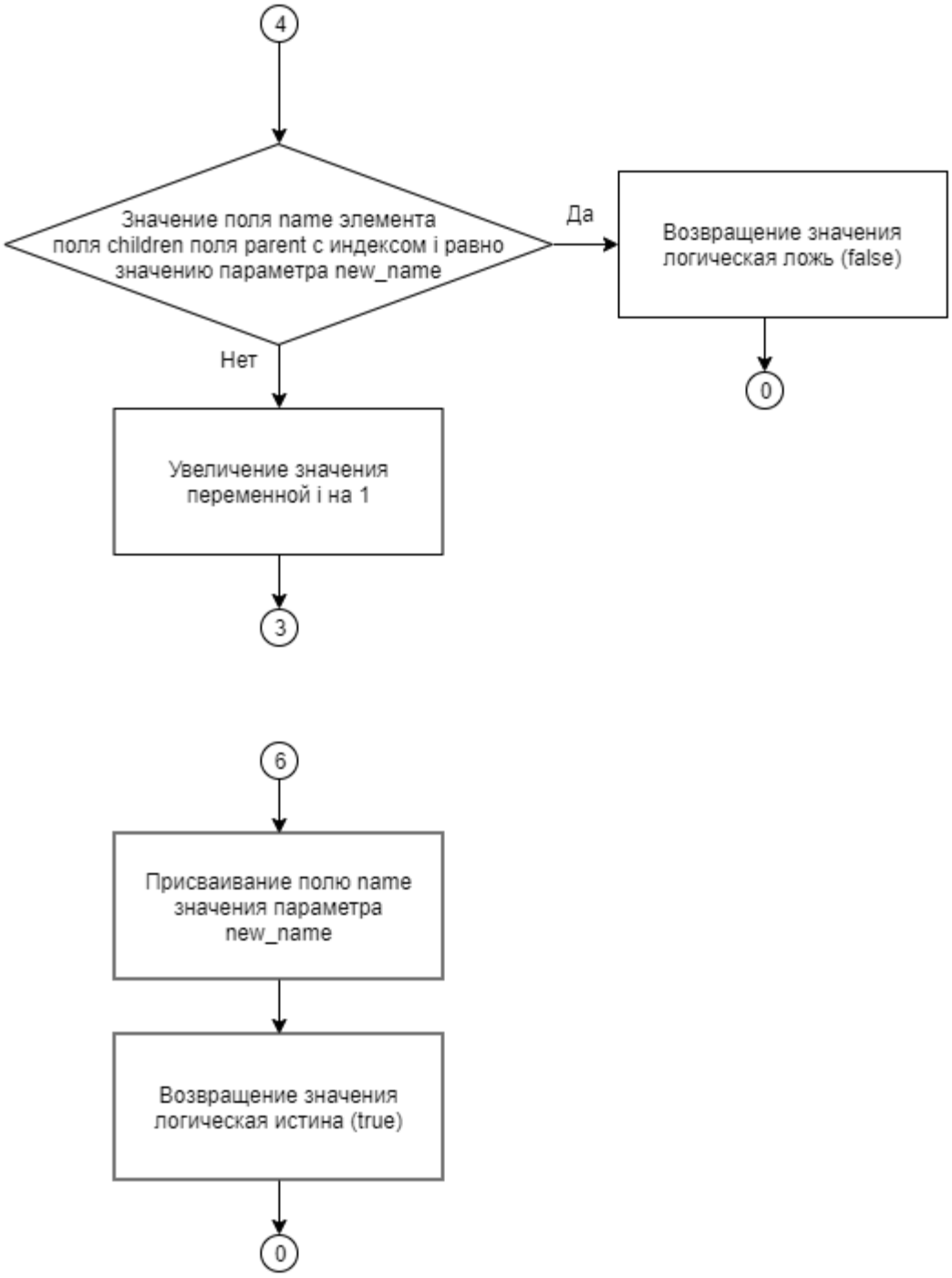

Рисунок 4 – Блок-схема алгоритма

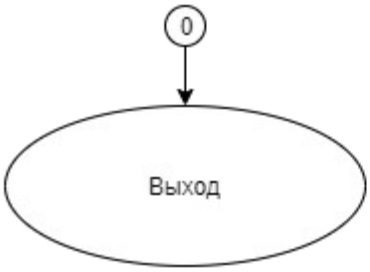

Рисунок 5 – Блок-схема алгоритма

Рисунок 6 – Блок-схема алгоритма

Рисунок 7 – Блок-схема алгоритма

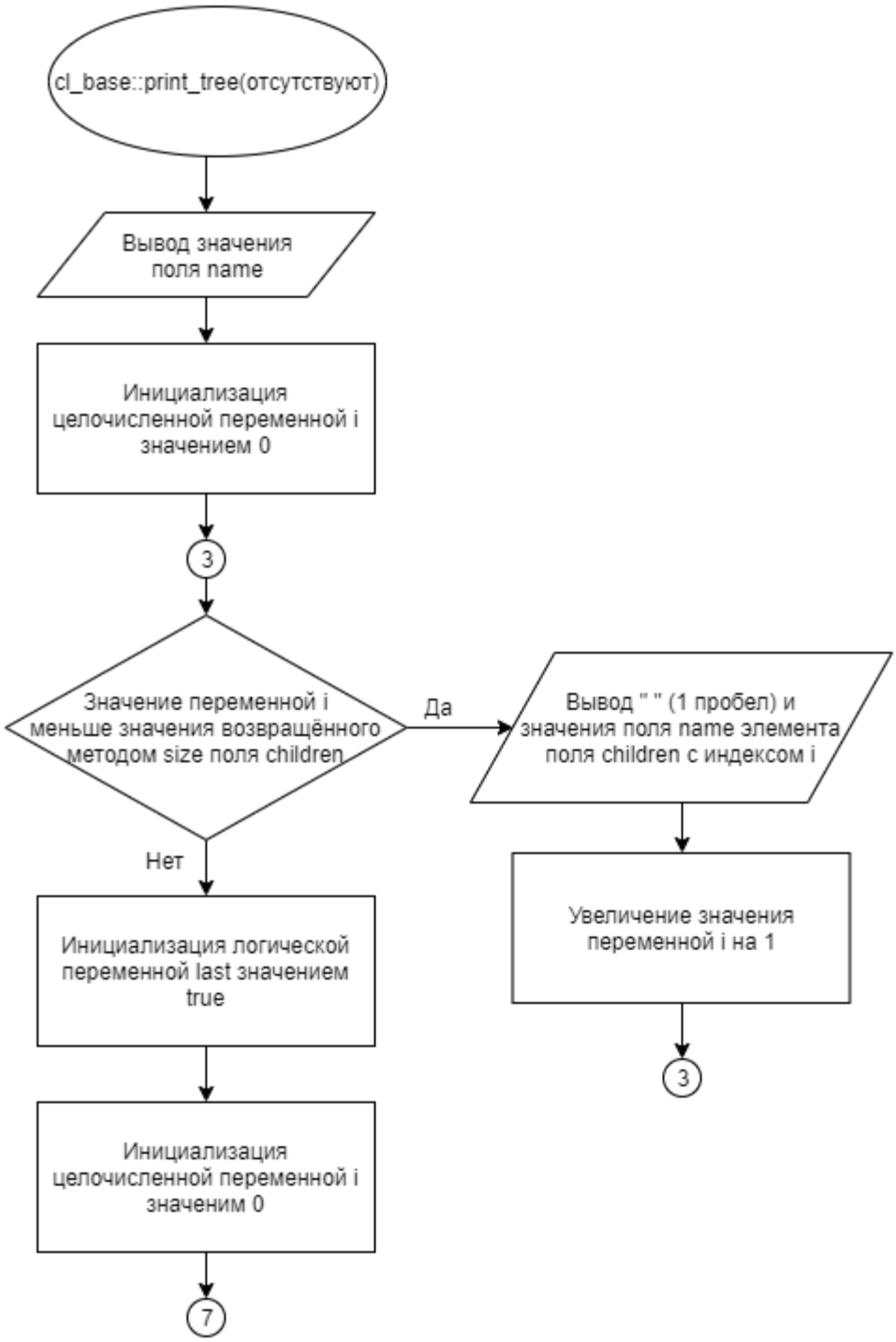

Рисунок 8 – Блок-схема алгоритма

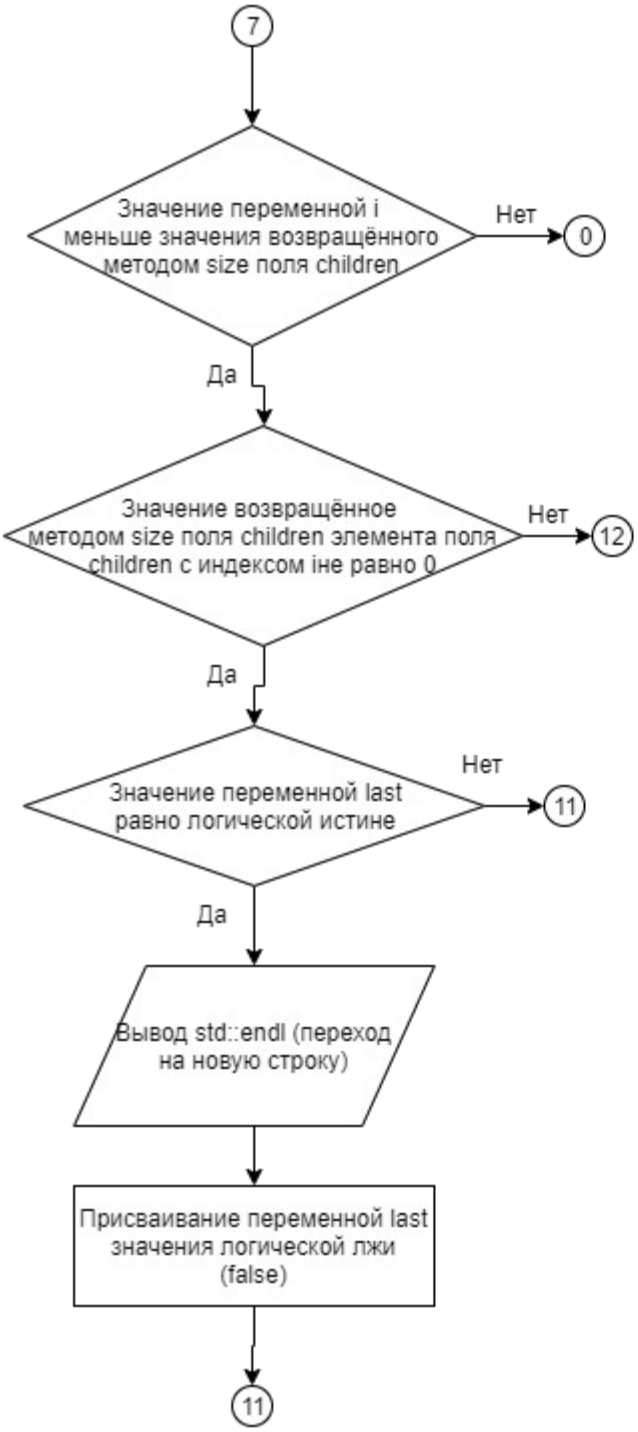

Рисунок 9 – Блок-схема алгоритма

Рисунок 10 – Блок-схема алгоритма

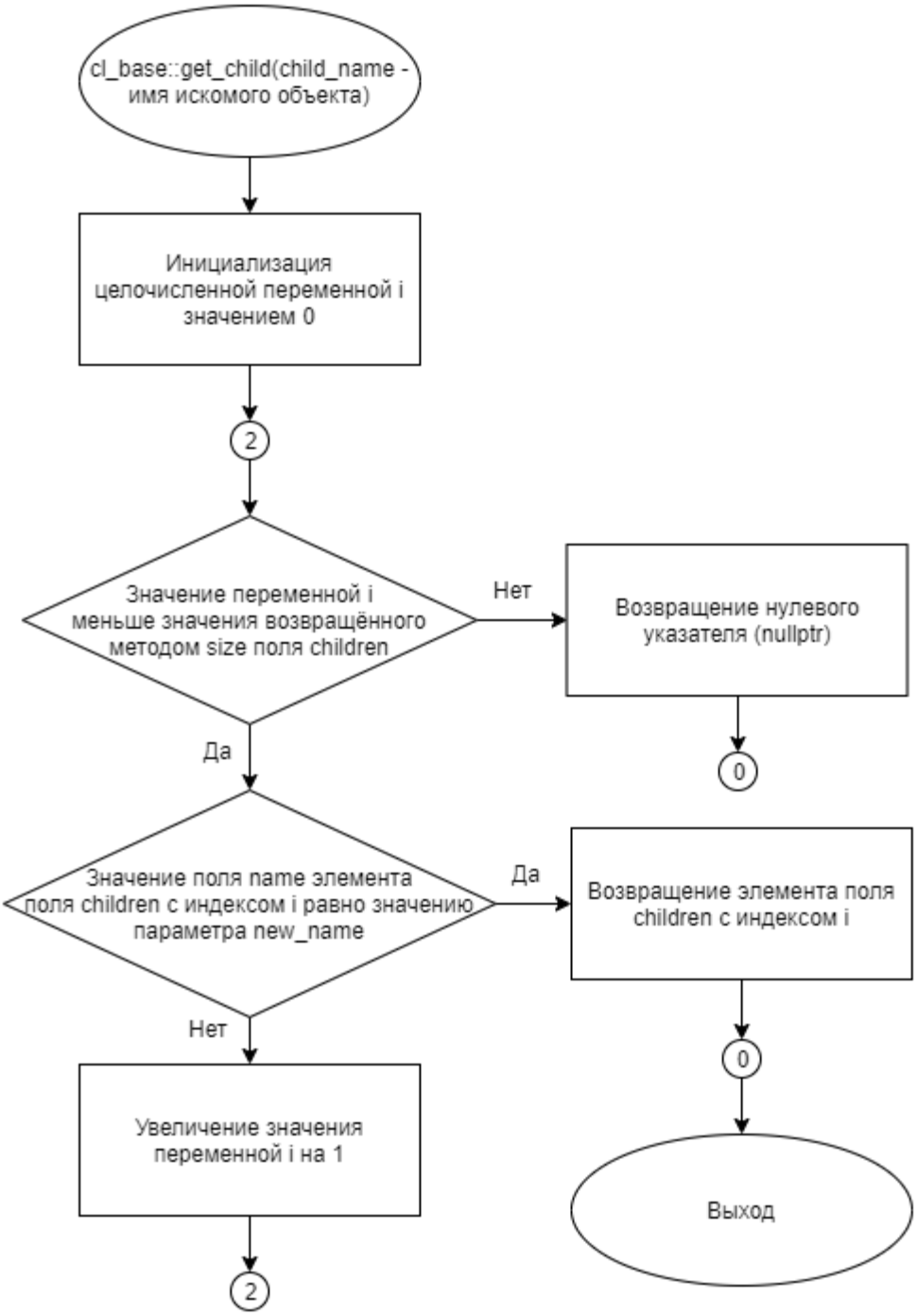

Рисунок 11 – Блок-схема алгоритма

Рисунок 12 – Блок-схема алгоритма

Рисунок 13 – Блок-схема алгоритма

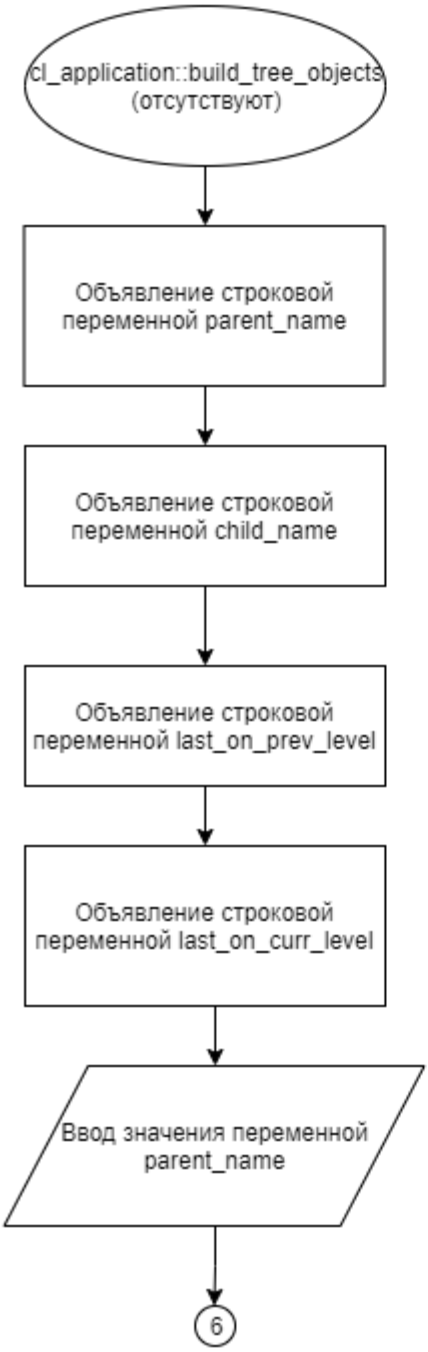

Рисунок 14 – Блок-схема алгоритма

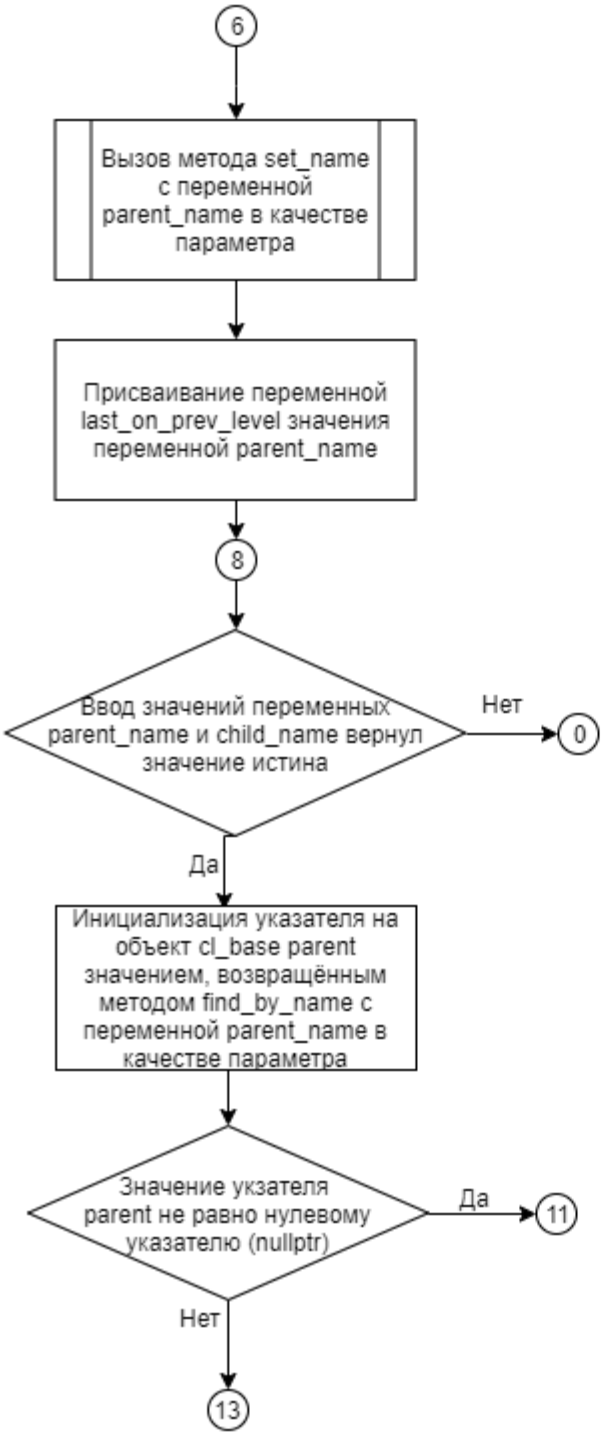

Рисунок 15 – Блок-схема алгоритма

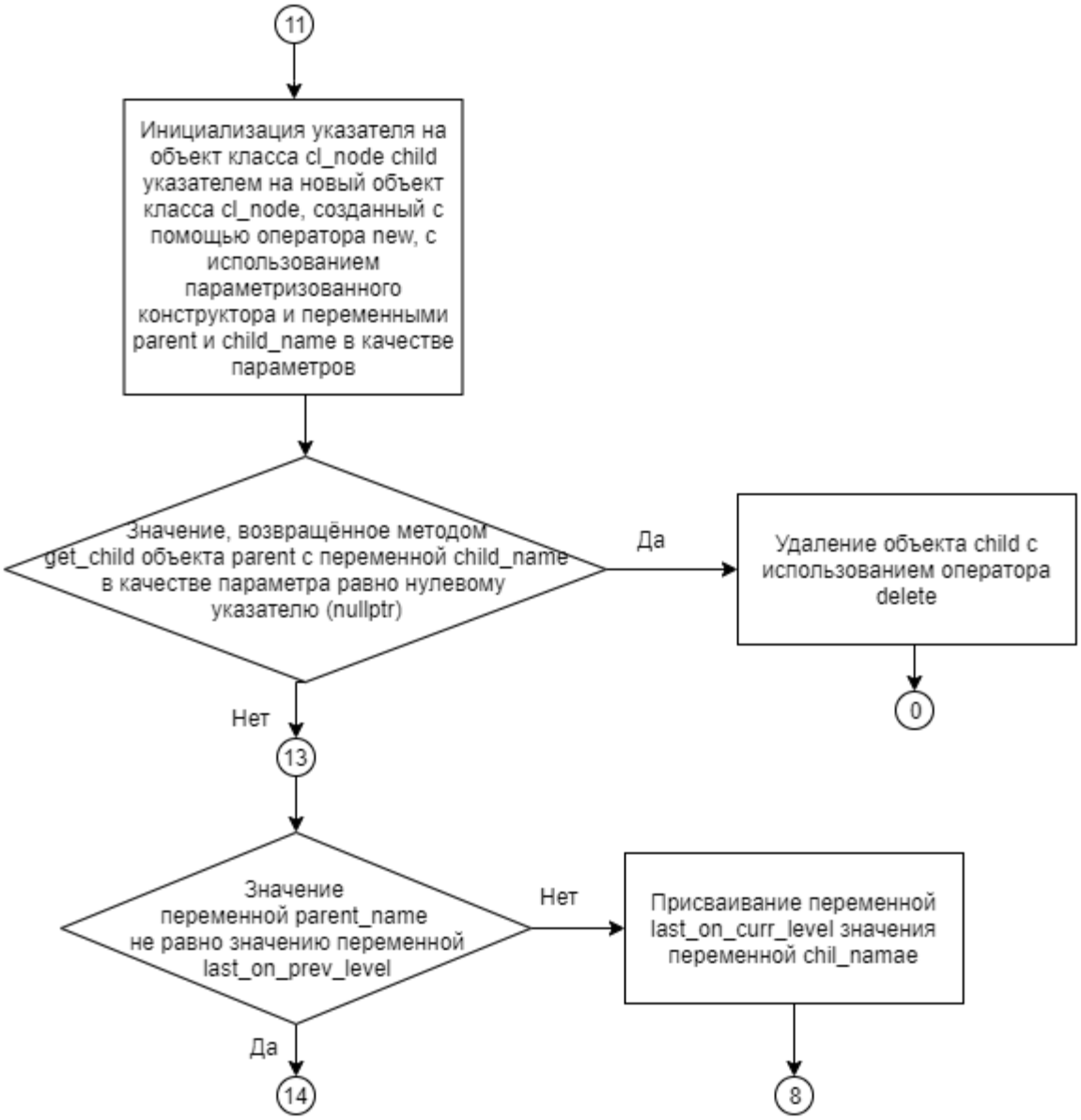

Рисунок 16 – Блок-схема алгоритма

Рисунок 17 – Блок-схема алгоритма

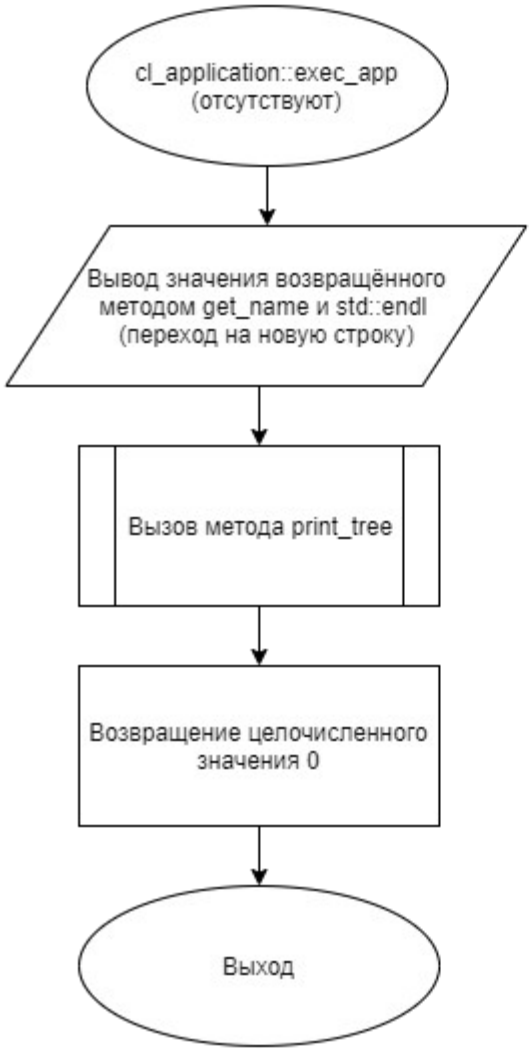

Рисунок 18 – Блок-схема алгоритма

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"

// Параметризованный конструктор
cl_application::cl_application(cl_application* parent) : cl_base(parent) {}

// Метод построения исходного дерева иерархии объектов
void cl_application::build_tree_objects()
{
	std::string parent_name;
	std::string child_name;

	// Указатель на последний объект,
	// которому был добавлен
	// подчинённый объект
	cl_base* last_parent = this;
	// Указатель на последий добавленный
	// подчинённый объект
	cl_base* last_child = this;

	std::cin >> parent_name;
	set_name(parent_name);

	// Пока успешно происходит ввод пары имён
	while (std::cin >> parent_name >> child_name)
	{
		if (parent_name == child_name)
		{
			break;
		}
		// Если введённое имя совпадает с имененем
		// последнего объекта которому был добавлен
		// подчинённый объект
		else if (parent_name == last_parent->get_name() &&
				last_parent->get_child(child_name) == nullptr)
		{
			// Ему добавляется новый
			last_child = new cl_node(last_parent, child_name);
		}
		// Если введённое имя совпадает с именем
		// последнего добавленного подчинённого объекта
		else if (parent_name == last_child->get_name() &&
```

```
                        last_child->get_child(child_name) == nullptr)
            {
                  // Изменяем указатель на последний объект,
                  // которому был доавлен подчинённый
                  // и добавляем ему подчинённый
                  last_parent = last_child;
                  last_child = new cl_node(last_parent, child_name);
            }
      }
}

// Метод запуска приложения
int cl_application::exec_app()
{
      // Вывод имени объекта
      std::cout << get_name() << std::endl;
      // Вызов метода вывода дерева иерархии
      print_tree();
      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"
#include "cl_node.h"
#include <iostream>
#include <string>

class cl_application : public cl_base
{
private:
public:

      // Параметризованный конструктор
      cl_application(cl_application* parent);

      // Метод построения исходного дерева иерархии объектов
      void build_tree_objects();

      // Метод запуска приложения
      int exec_app();
};

#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"

// Параметризированный конструктор
cl_base::cl_base(cl_base* parent, std::string name)
{
	// Установка родительского объекта
	this->parent = parent;
	// Установка имени объекта
	this->name = name;

	if (parent != nullptr)
	{
		// Добавление объекта к подчинённым
		// объектом родительского объекта
		parent->children.push_back(this);
	}
}

// Метод редактирования имени объекта
bool cl_base::set_name(std::string new_name)
{
	if (parent != nullptr)
	{
		for (int i = 0; i < parent->children.size(); i++)
		{
			if (parent->children[i]->name == new_name)
			{
				// Объект с имененм, совпадающим
				// с новым именем найден среди
				// подчинённых объектов родительского
				return false;
			}
		}
	}

	// Установка нового имени объекта
	this->name = new_name;
	return true;
}

// Метод получения имени объекта
std::string cl_base::get_name()
{
	return name;
}

// Метод получения указателя на головной объект текущего объекта
cl_base* cl_base::get_parent()
{
	return parent;
}
```

```
// Метод вывода наименований объектов в дереве иерархии слева
// направо и сверху вниз
void cl_base::print_tree()
{
	// Вывод имени текущего объекта
	std::cout << name;
	// Вывод имён всех подчинённых объектов
	for (int i = 0; i < children.size(); i++)
	{
		std::cout << "  " << children[i]->name;
	}

	// Рекурсивный вызов метода print_tree
	// для всех подчинённых объектов, у
	// которых есть свои подчинённые объекты
	bool last = true;
	for (int i = 0; i < children.size(); i++)
	{
		if (children[i]->children.size() != 0)
		{
			if (last)
			{
				std::cout << std::endl;
				last = false;
			}
			children[i]->print_tree();
		}
	}
}

// Метод получения указателя на непосредственно подчиненный
// объект по его имени
cl_base* cl_base::get_child(std::string child_name)
{
	for (int i = 0; i < children.size(); i++)
	{
		if (children[i]->name == child_name)
		{
			// Среди подчинённых объектов
			// найден объект с именем,
			// совпадающим с искомым
			return children[i];
		}
	}

	// Объект не найден
	return nullptr;
}
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H

#include <iostream>
#include <string>
#include <vector>

class cl_base
{
private:

	// Наименование объекта
	std::string name;

	// Указатель на головной объект для текущего объекта
	cl_base* parent;

	// Динамический массив указателей на объекты,
	// подчиненные к текущему объекту в дереве иерархии
	std::vector<cl_base*> children;

public:

	// Параметризированный конструктор
	cl_base(cl_base* parent, std::string name="");

	// Метод редактирования имени объекта
	bool set_name(std::string new_name);

	// Метод получения имени объекта
	std::string get_name();

	// Метод получения указателя на головной объект текущего объекта
	cl_base* get_parent();

	// Метод вывода наименований объектов в дереве иерархии слева
	// направо и сверху вниз
	void print_tree();

	// Метод получения указателя на непосредственно подчиненный
	// объект по его имени
	cl_base* get_child(std::string child_name);
};

#endif
```

5.5 Файл cl_node.cpp

Листинг 5 – cl_node.cpp

```
#include "cl_node.h"

cl_node::cl_node(cl_base* parent, std::string name)
	: cl_base(parent, name) {}
```

5.6 Файл cl_node.h

Листинг 6 – cl_node.h

```
#ifndef CL_NODE_H
#define CL_NODE_H

#include "cl_base.h"
#include <string>

class cl_node : public cl_base
{
private:
public:
	// Параметризованный конструктор
	cl_node(cl_base* parent, std::string name);
};

#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
#include "cl_application.h"

int main()
{
	cl_application ob_cl_application(nullptr);
	ob_cl_application.build_tree_objects();
	return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 13.

Таблица 13 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5

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).