Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:

- метод поиска объекта на ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
- метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется. При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

1. Построить дерево объектов системы (в методе корневого объекта построения исходного дерева объектов).

2. В методе корневого объекта запуска моделируемой системы реализовать:

2.1 Вывод на консоль иерархического дерева объектов в следующем виде:

```
root
    ob_1
        ob_2
    ob_3
        ob_4
            ob_5
        ob_6
            ob_7
```

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам).

2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

```
root  is ready
    ob_1  is ready
        ob_2  is ready
    ob_3  is ready
        ob_4 is not ready
            ob_5 is not ready
        ob_6 is ready
            ob_7 is not ready
```

1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка

«Наименование корневого объекта»

Со второй строки

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

. . . . .

endtree

Со следующей строки вводятся команды включения или отключения объектов

«Наименование объекта» «Номер состояния объекта»

Пример ввода

```
app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
```

1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

```
Object tree
«Наименование корневого объекта»
    «Наименование объекта 1»
        «Наименование объекта 2»
    «Наименование объекта 3»
. . . . .
The tree of objects and their readiness
«Наименование корневого объекта» «Отметка готовности»
    «Наименование объекта 1» «Отметка готовности»
        «Наименование объекта 2» «Отметка готовности»
    «Наименование объекта 3» «Отметка готовности»
. . . . .
«Отметка готовности» - равно «is ready» или «is not ready»
```

Отступ каждого уровня иерархии 4 позиции.

Пример вывода

```
Object tree
app_root
    object_01
        object_07
    object_02
        object_04
        object_05
The tree of objects and their readiness
app_root is ready
    object_01 is ready
        object_07 is not ready
    object_02 is ready
        object_04 is ready
        object_05 is not ready
```

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_application, cl_2, cl_3, cl_4, cl_5 и cl_6.

Иерархия классов представленна в таблице 1.

- Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	cl_base			Базовый класс в иерархии классов. Содержит основн		

				ые поля и методы.		
		cl_application	public		2	
		cl_2	public		3	
		cl_3	public		4	
		cl_4	public		5	
		cl_5	public		6	
		cl_6	public		7	
2	cl_application			Класс корневого объекта (приложения)		
3	cl_2			Класс объекта дерева		
4	cl_3			Класс объекта дерева		
5	cl_4			Класс объекта дерева		
6	cl_5			Класс объекта дерева		
7	cl_6			Класс		

				объекта дерева		

1. Класс cl_base:
    - Свойства (поля):
        - Поле, отвечающее за состояние объекта:
            - Наименование - state;
            - Тип - целочисленный;
            - Модификатор доступа - закрытый;
        - 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 за хранение указателей на подчинённые объекты в дереве иерархии:
            - Наименование - children;
            - Тип - вектор указателей на объекты класса cl_base;
            - Модификатор доступа - закрытый;
    - Методы:
        - Метод print_tree:
            - Функционал - вывод дерева иерархии объектов;
            - Возвращаемое значение - отсутствует (void);

        - Модификатор доступа - открытый
        - Параметры:
            - prefix - строка, выводящаяся в начале вывода;
- Метод find_in_tree:
    - Функционал - поиск объекта в дереве иерархии объектов по указанному имени;
    - Возвращаемое значение - указатель на объект cl_base;
    - Модификатор доступа - открытый
    - Параметры:
        - wanted_name - имя искомого объекта (строка);
- Метод print_tree_with_state:
    - Функционал - вывод дерева иерархии объектов и отметок о готовности объектов;
    - Возвращаемое значение - отсутствует (void);
    - Модификатор доступа - открытый;
    - Параметры:
        - prefix - строка, выводящаяся в начале вывода;
- Метод get_state:
    - Функционал - возвращение состояния объекта;
    - Возвращаемое значение - целочисленный код состояния объекта;
    - Модификатор доступа - открытый;
    - Параметры - отсутствуют;
- Метод set_state:
    - Функционал - устновка состояния объекта;
    - Возвращаемое значение - отсутствует (void);
    - Модификатор доступа - открытый;

        - Параметры:
            - state - новое состояние (целое число);
    - Метод get_root:
        - Функционал - возвращение состояния объекта;
        - Возвращаемое значение - указатель на объект класса cl_base;
        - Модификатор доступа - открытый;
        - Параметры - отсутствуют;
    - Метод find_in_branch:
        - Функционал - поиск объекта в дереве иерархии объектов от корня дерева;
        - Возвращаемое значение - указатель на объект cl_base;
        - Модификатор доступа - открытый;
        - Параметры:
            - wanted_name - имя искомого объекта (строка).

2. Класс cl_application:
    - Методы:
        - Метод build_tree_objects:
            - Функционал - создание исходного дерева иерархии объектов;
            - Возвращаемое значение - отсутствует (void);
            - Модификатор доступа - открытый;
            - Параметры - отсутствуют.
        - Метод exec_app:
            - Функционал - запуск приложения;
            - Возвращаемое значение - целочисленный код завершения работы приожения ;

        - Модификатор доступа - открытый;
        - Параметры - отсутствуют;
    - Метод read_states:
        - Функционал - считывание имён объектов и их состояний и устновка готовности этим объектам;
        - Возвращаемое значение - отсутствует (void);
        - Модификатор доступа - открытый;
        - Параметры - отсутствуют.
3. Класс cl_2:
    - Методы:
        - Параметризованный конструктор:
            - Функционал - создание объекта класса с заданным именем и головным объектом;
            - Возвращаемое значение - объект класса cl_2;
            - Модификатор доступа - открытый;
            - Параметры:
                - parent - указатель на головной объект (указатель на объект класса cl_base);
                - name - имя объекта (строка).
4. Класс cl_3:
    - Методы:
        - Параметризованный конструктор:
            - Функционал - создание объекта класса с заданным именем и головным объектом;
            - Возвращаемое значение - объект класса cl_3;
            - Модификатор доступа - открытый;
            - Параметры:

        - parent - указатель на головной объект (указатель на объект класса cl_base);
        - name - имя объекта (строка).

5. Класс cl_4:
    - Методы:
        - Параметризованный конструктор:
            - Функционал - создание объекта класса с заданным именем и головным объектом;
            - Возвращаемое значение - объект класса cl_4;
            - Модификатор доступа - открытый;
            - Параметры:
                - parent - указатель на головной объект (указатель на объект класса cl_base);
                - name - имя объекта (строка).
6. Класс cl_5:
    - Методы:
        - Параметризованный конструктор:
            - Функционал - создание объекта класса с заданным именем и головным объектом;
            - Возвращаемое значение - объект класса cl_5;
            - Модификатор доступа - открытый;
            - Параметры:
                - parent - указатель на головной объект (указатель на объект класса cl_base);
                - name - имя объекта (строка).
7. Класс cl_6:
    - Методы:

- Параметризованный конструктор:
    - Функционал - создание объекта класса с заданным именем и головным объектом;
    - Возвращаемое значение - объект класса cl_6;
    - Модификатор доступа - открытый;
    - Параметры:
        - parent - указатель на головной объект (указатель на объект класса cl_base);
        - name - имя объекта (строка).

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: отсутствуют.

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		Возвращение значения, возвращённого методом exec_app класса ob_cl_application	∅

3.2 Алгоритм метода print_tree класса cl_base

Функционал: вывод дерева иерархии объектов.

Параметры: prefix - строка, выводящаяся в начале вывода.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1	Значение поля parent равно нулевому указателю (nullptr)	Вывод значения поля name	2
			2
2		Инициализация целочисленной переменной i значеним 0	3
3	Значение переменной i меньше значения, возвращённого методом size поля children		4
			∅
4		Вывод "\n" (переход на новую строку), значения переменной prefix и поля name элемента с индексом i поля children	5
5		Вызов метода print_tree элемента с индексом i поля children с суммой строк prefix и " " (4 пробела) в качестве аргумента	6
6		Увеличение значения переменной i на 1	3

3.3 Алгоритм метода find_in_tree класса cl_base

Функционал: поиск объекта в дереве иерархии объектов с заданным именем.

Параметры: wanted_name - имя искомого объекта.

Возвращаемое значение: указатель на объект cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода find_in_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя на объект cl_base res	2

№	Предикат	Действия	№ перехода
		значением nullptr (нулевой указатель)	
2	Значение поля name равно значению перменной wanted_name	Присваивание указателю res указателя на текущий объект (this)	3
			3
3		Инициализация целочисленной переменной i значением 0	4
4	Значение переменной i меньше значения, возвращённого методом size поля children		5
			9
5		Инициализация указателя на объект cl_base found значением, возвращённого методом find_in_tree элемента с индексом i поля children с переменной wanted_name в качестве аргумента	6
6	Значение указателя found не равно нулевому указателю (nullptr)		7
			8
7	Значение указателя res равно нулевому указателю (nullptr)	Присваивание указателю res значения указателя found	8
		Возврат нулевого указателя (nullptr)	∅
8		Увеличение значения переменной i на 1	4
9		Возврат указателя res	∅

3.4 Алгоритм метода print_tree_with_state класса cl_base

Функционал: вывод дерева иерархии объектов и отметок о готовности

объектов.

Параметры: prefix - строка, выводящаяся в начале вывода.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода print_tree_with_state класса cl_base

№	Предикат	Действия	№ перехода
1	Значение поля parent равно нулевому указателю (nullptr)	Вывод значения поля name, строки " is not ready" в случае, если значение поля state равно 0, и строки " is ready" в другом случае	2
			2
2		Инициализация целочисленной переменной i значением 0	3
3	Значение переменной i меньше значения, возвращённого методом size поля children		4
			∅
4		Вывод "\n" (переход на новую строку), значения переменной prefix, значения, возарщённого методом get_name элемента с индексом i поля children и строки " is not ready" в случае, если значение значени, возвращённое методом get_state элемента с индексом i поля children равно 0, и строки " is ready" в другом случае	5
5		Вызов метода print_tree_with_state элемента с индексом i поля children с суммой строк prefix и " " (4 пробела) в качестве аргумента	6
6		Увеличение значения переменной i на 1	3

3.5 Алгоритм метода get_state класса cl_base

Функционал: возвращение состояния объекта.

Параметры: отсутствуют.

Возвращаемое значение: целочисленное значение состояния объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_state класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля state	∅

3.6 Алгоритм метода set_state класса cl_base

Функционал: устновка состояния объекта.

Параметры: state - новое состояние (целое число).

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_state класса cl_base

№	Предикат	Действия	№ перехода
1	Значение поля parent равно нулевому указателю (nullptr) или поле state объекта parent не равно 0	Присваивание полю state значения переменной (аргумента) state	2
		Присваивание полю state значения 0	2
2	Значение поля state равно 0		3
			∅
3		Инициализация целочисленной переменной i значением 0	4
4	Значение переменной i		5

№	Предикат	Действия	№ перехода
	меньше значения, возвращённого методом size поля children		
			∅
5		Вызов метода set_state элемента с индексом i поля children с значением 0 в качестве аргумента	6
6		Увеличение значения переменной i на 1	4

3.7 Алгоритм метода read_states класса cl_application

Функционал: считывание имён объектов и их состояний и устновка готовности этим объектам.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода read_states класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление строковой переменной input_name	2
2		Инициализцаия целочисленной переменной input_state значением 0	3
3	Ввод значений переменных input_name и input_state вернул значение истина		4
			∅
4		Инициализация указателя на объект класса cl_base obj значением, возвращённым методом find_in_tree со значением input_name в качестве аргумента	5
5	Значение указателя obj не	Вызов метода set_state объекта obj со значением	3

№	Предикат	Действия	№ перехода
	равно нулевому указателю (nullptr)	переменной input_state в качестве параметра	
			3

3.8 Алгоритм метода build_tree_objects класса cl_application

Функционал: создание исходного дерева иерархии объектов.

Параметры: отсутствуют..

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – 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 целочисленной переменной class_number значением 0	4
4		Инициализация указателя на объект cl_base input_parent значением nullpptr (нулевой указатель)	5
5		Инициализация указателя на объект cl_base input_child значением nullpptr (нулевой указатель)	6
6		Ввод переменной parent_name	7
7		Вызов метода set_name со значением переменной parent_name в качестве аргумента	8
8		Ввод значения переменной parent_name	9
9	Значение переменной parent_name равно "endtree"		∅
			10

№	Предикат	Действия	№ перехода
10		Ввод значений переменных child_name и class_number	11
11		Присваивание указателю input_parent значения, возвращённого методом find_in_tree со значением переменной parent_name в качестве параметра	12
12	Значение указателя input_parent не равно нулевому указателю (nullptr) и значение, возвращённое методом get_child объекта input_parent со значением переменной child_name в качестве параметра, равно нулевому указателю (nullptr)		13
			8
13	Значение переменной class_number равно 2	Присваивание указателю input_child указателя на новый объект класса cl_2, созданный с помощью оператора new, с использованием параметризованного конструктора и значениями переменных input_parent и child_name в качестве параметров	8
			14
14	Значение переменной class_number равно 3	Присваивание указателю input_child указателя на новый объект класса cl_3, созданный с помощью оператора new, с использованием параметризованного конструктора и значениями переменных input_parent и child_name в качестве параметров	8
			15
1	Значение переменной	Присваивание указателю input_child указателя на	8

№	Предикат	Действия	№ перехода
5	class_number равно 4	новый объект класса cl_4, созданный с помощью оператора new, с использованием параметризованного конструктора и значениями переменных input_parent и child_name в качестве параметров	
			16
16	Значение переменной class_number равно 5	Присваивание указателю input_child указателя на новый объект класса cl_5, созданный с помощью оператора new, с использованием параметризованного конструктора и значениями переменных input_parent и child_name в качестве параметров	8
			17
17	Значение переменной class_number равно 6	Присваивание указателю input_child указателя на новый объект класса cl_6, созданный с помощью оператора new, с использованием параметризованного конструктора и значениями переменных input_parent и child_name в качестве параметров	8
			8

3.9 Алгоритм метода exec_app класса cl_application

Функционал: запуск приложения.

Параметры: отсутствуют.

Возвращаемое значение: целочисленный код завершения работы приложения.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод строки "Object tree\n"	2
2		Вызов метода print_tree со строкой " " (4 пробела) в качестве аргумента	3
3		Вывод строки "\nThe tree of objects and their readiness\n"	4
4		Вызов метода read_states	5
5		Вызов метода print_tree_with_states со строкой " " (4 пробела) в качестве аргумента	6
6		Возврат 0	∅

3.10 Алгоритм конструктора класса cl_2

Функционал: создание объекта класса с заданным именем и головным объектом.

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

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса cl_base со значениеями parent и name в качестве параметров	∅

3.11 Алгоритм конструктора класса cl_3

Функционал: создание объекта класса с заданным именем и головным объектом.

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

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса cl_base со значениеями parent и name в качестве параметров	∅

3.12 Алгоритм конструктора класса cl_4

Функционал: создание объекта класса с заданным именем и головным объектом.

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

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса cl_base со значениеями parent и name в качестве параметров	∅

3.13 Алгоритм конструктора класса cl_5

Функционал: создание объекта класса с заданным именем и головным объектом.

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

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса cl_base со значениеями parent и name в качестве параметров	∅

3.14 Алгоритм конструктора класса cl_6

Функционал: создание объекта класса с заданным именем и головным объектом.

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

Алгоритм конструктора представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса cl_base со значениеями parent и name в качестве параметров	∅

3.15 Алгоритм метода get_root класса cl_base

Функционал: возвращает указатель на корень дерева иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода get_root класса cl_base

№	Предикат	Действия	№ перехода
1	Значение поля parent равно нулевому указателю (nullptr)	Возврат указателя на текущий объект (this)	∅
		Возврат значения, возвращённого методом get_root поля parent	∅

3.16 Алгоритм метода find_in_branch класса cl_base

Функционал: поиск объекта в дереве иерархии объектов от корня дерева.

Параметры: wanted_name - имя искомого объекта (строка).

Возвращаемое значение: указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода find_in_branch класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной amount значением возвращённым методом count_in_tree объекта, возвращённого методом get_root, с значением wanted_name в качестве параметра	2
2	Значение переменной amount равно 1	Возврат значения, возвращённого методом find_in_tree объетка, возвращённого методом get_root со значением wanted_name в качестве параметра	∅
		Возврат нулевого указателя (nullptr)	∅

3.17 Алгоритм метода count_in_tree класса cl_base

Функционал: подсчёт количества объектов с заданным именем в дереве иерархии.

Параметры: wanted_name - имя искомого объекта.

Возвращаемое значение: количество объекта с заданным именем (целое число).

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода count_in_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной res значением 0	2
2	Значение поля name равно значению переменной	Увеличение значения переменной res на 1	3

№	Предикат	Действия	№ перехода
	wanted_name		
			3
3		Инициализация целочисленной переменной i значением 0	4
4	Значение переменной i меньше значения, возвращённого методом size поля children		5
			7
5		Присваивание переменной res суммы значения переменной res и значения, возвращённого методом count_in_tree элемента поля children с индексом i со значением wanted_name в качестве параметра	6
6		Увеличение значения переменной i на 1	4
7		Возврат значения res	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-23.

Рисунок 1 – Блок-схема алгоритма

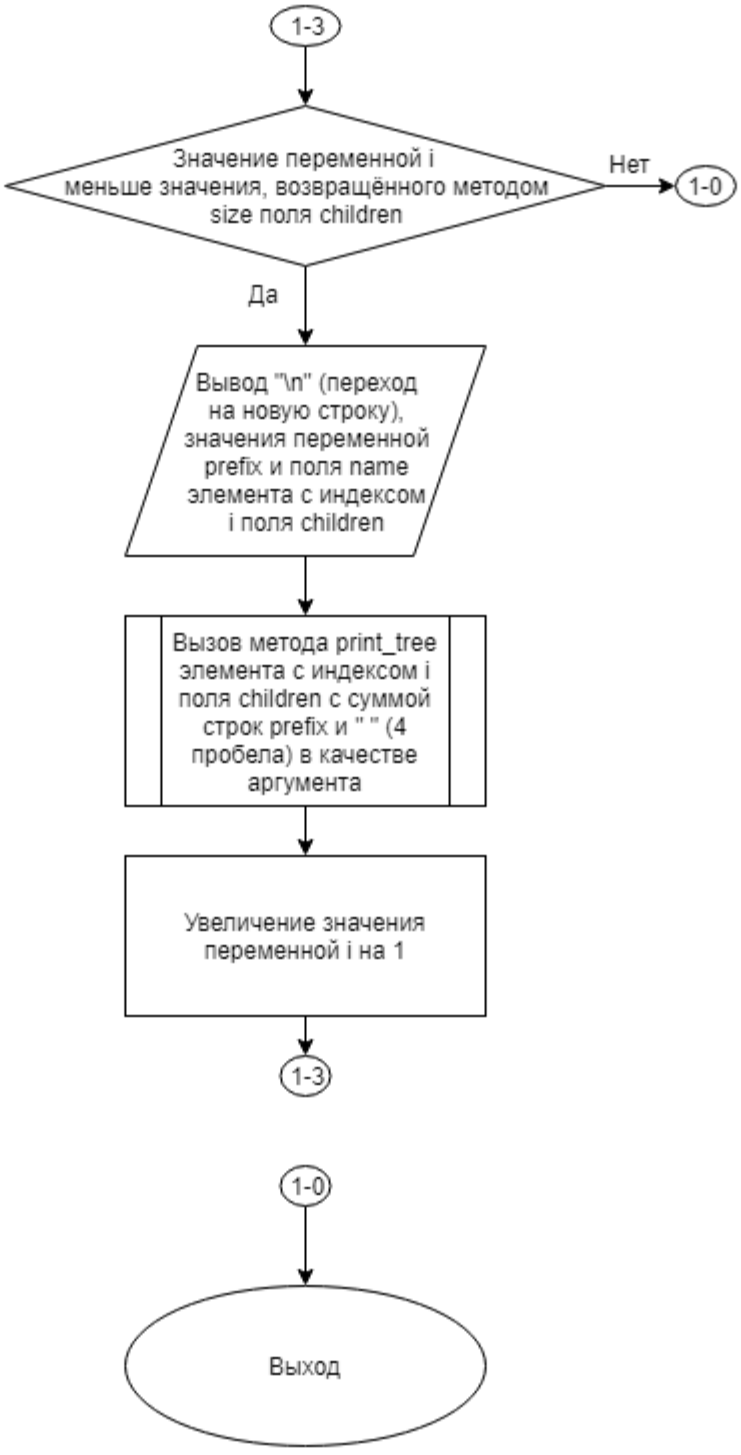

Рисунок 2 – Блок-схема алгоритма

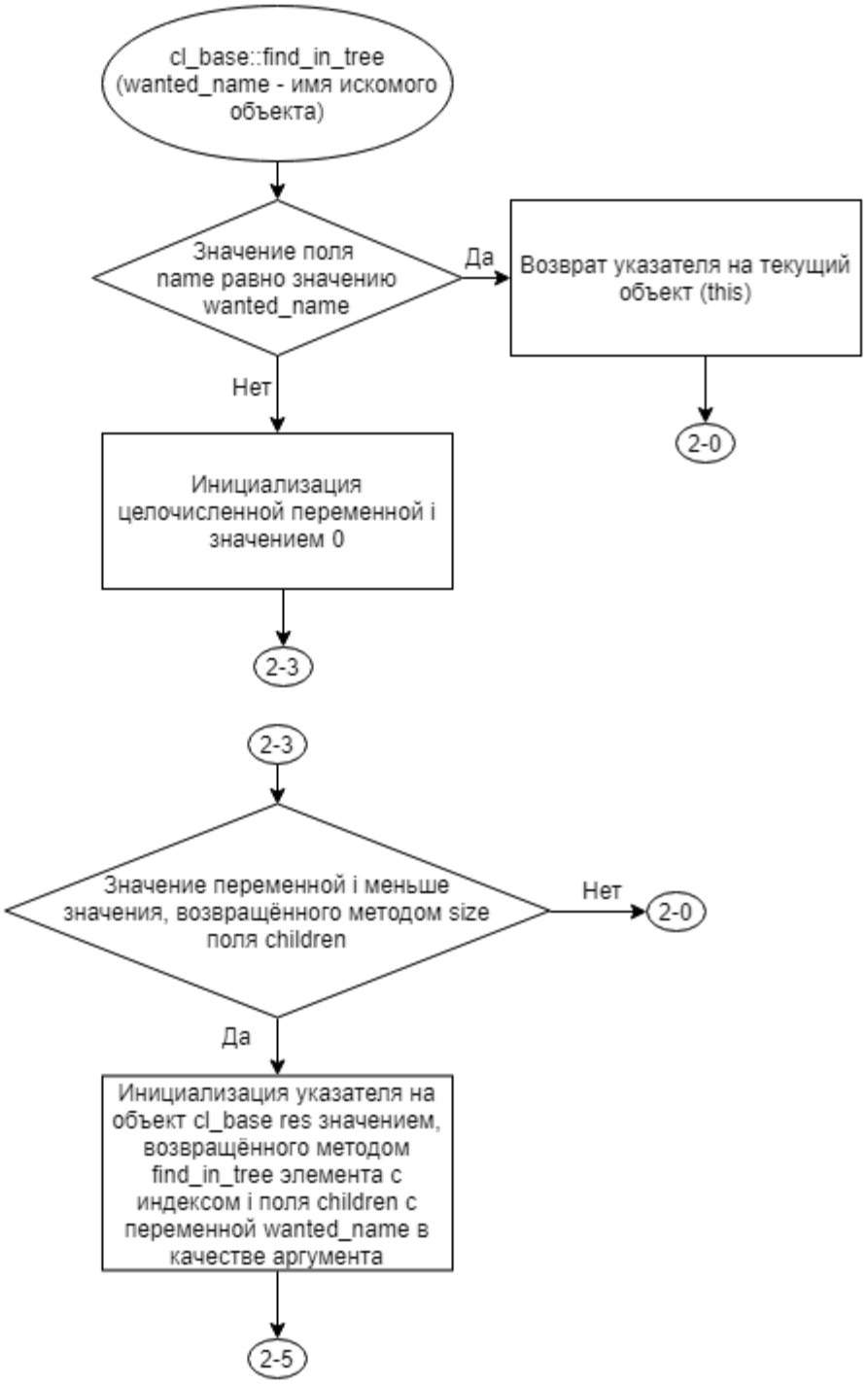

Рисунок 3 – Блок-схема алгоритма

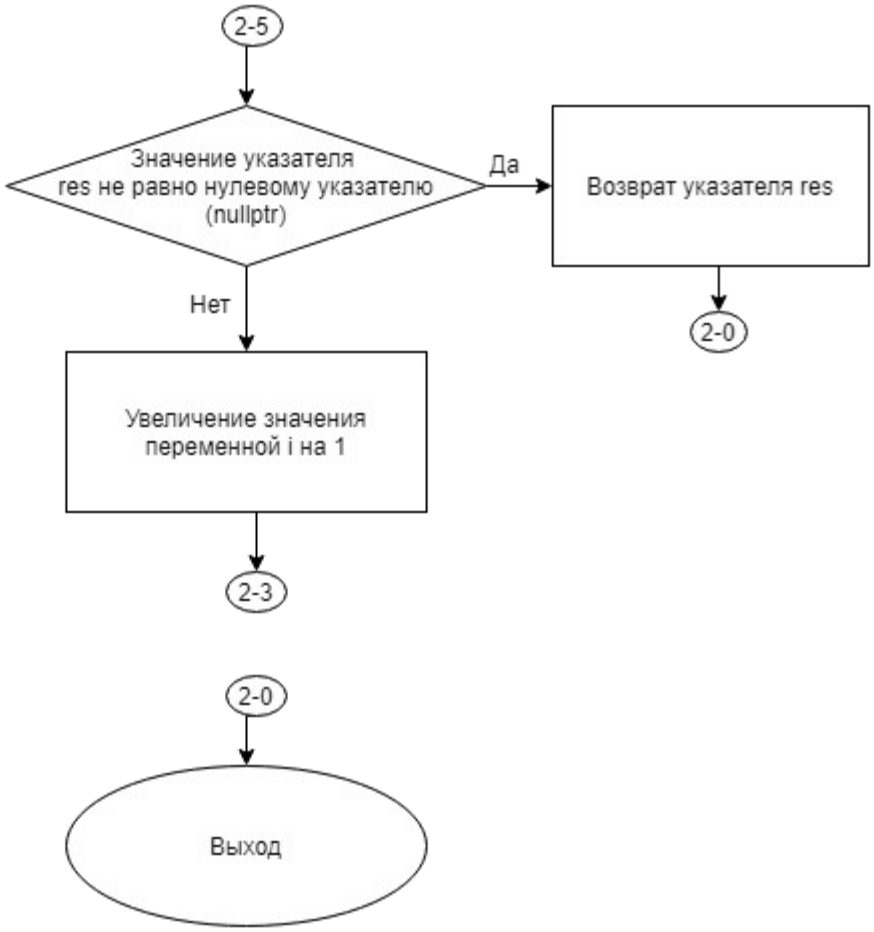

Рисунок 4 – Блок-схема алгоритма

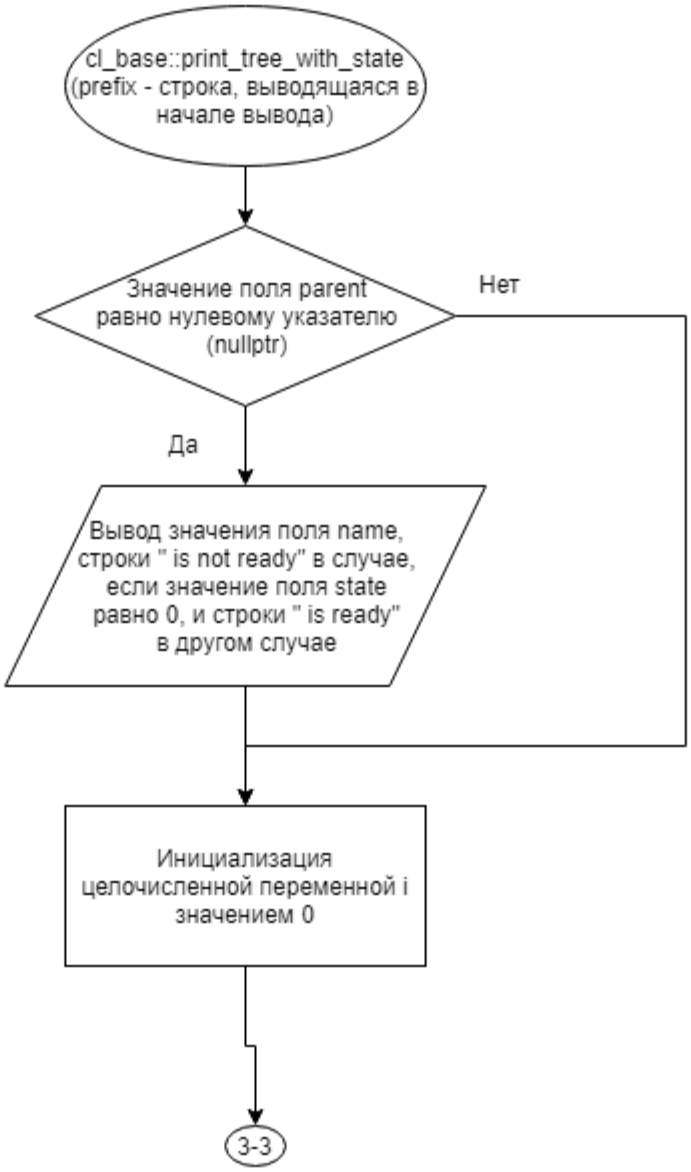

Рисунок 5 – Блок-схема алгоритма

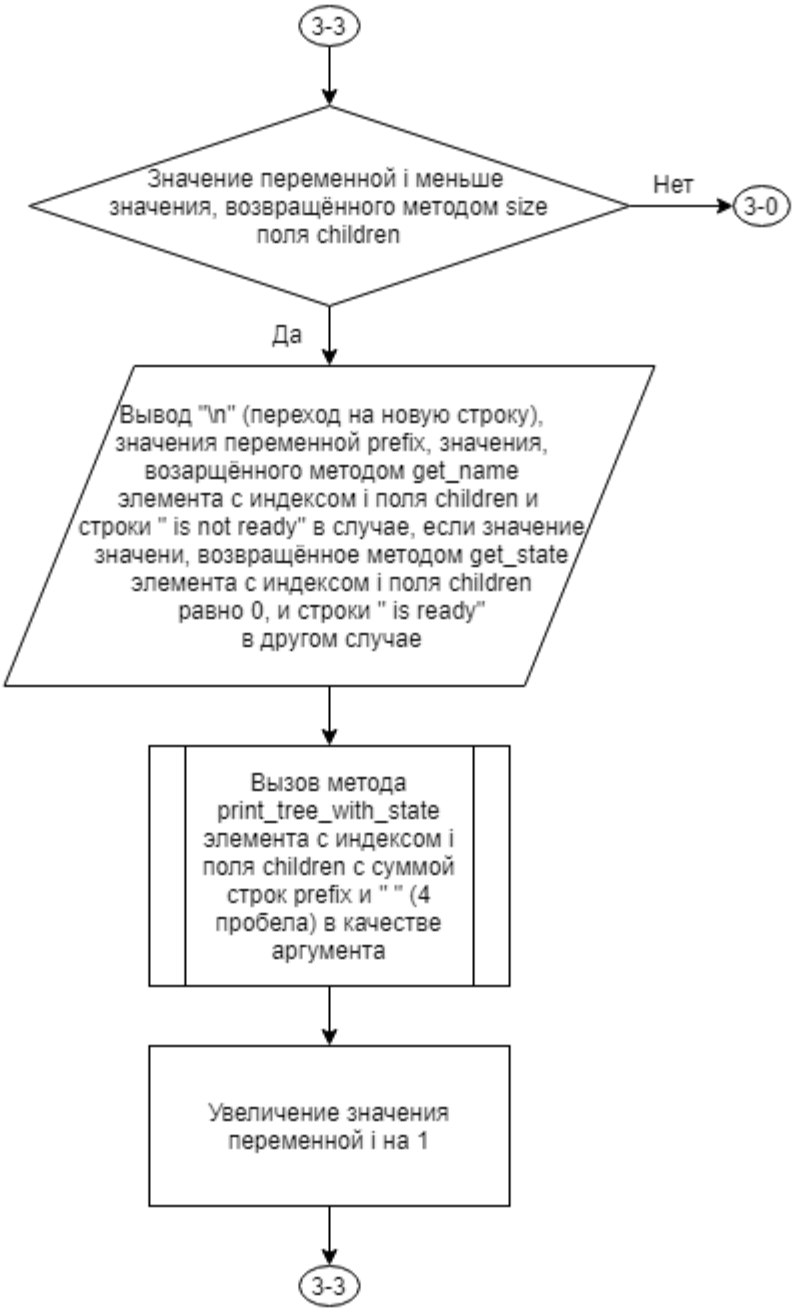

Рисунок 6 – Блок-схема алгоритма

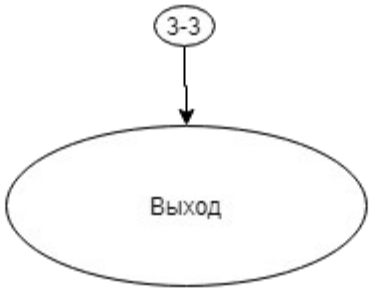

Рисунок 7 – Блок-схема алгоритма

Рисунок 8 – Блок-схема алгоритма

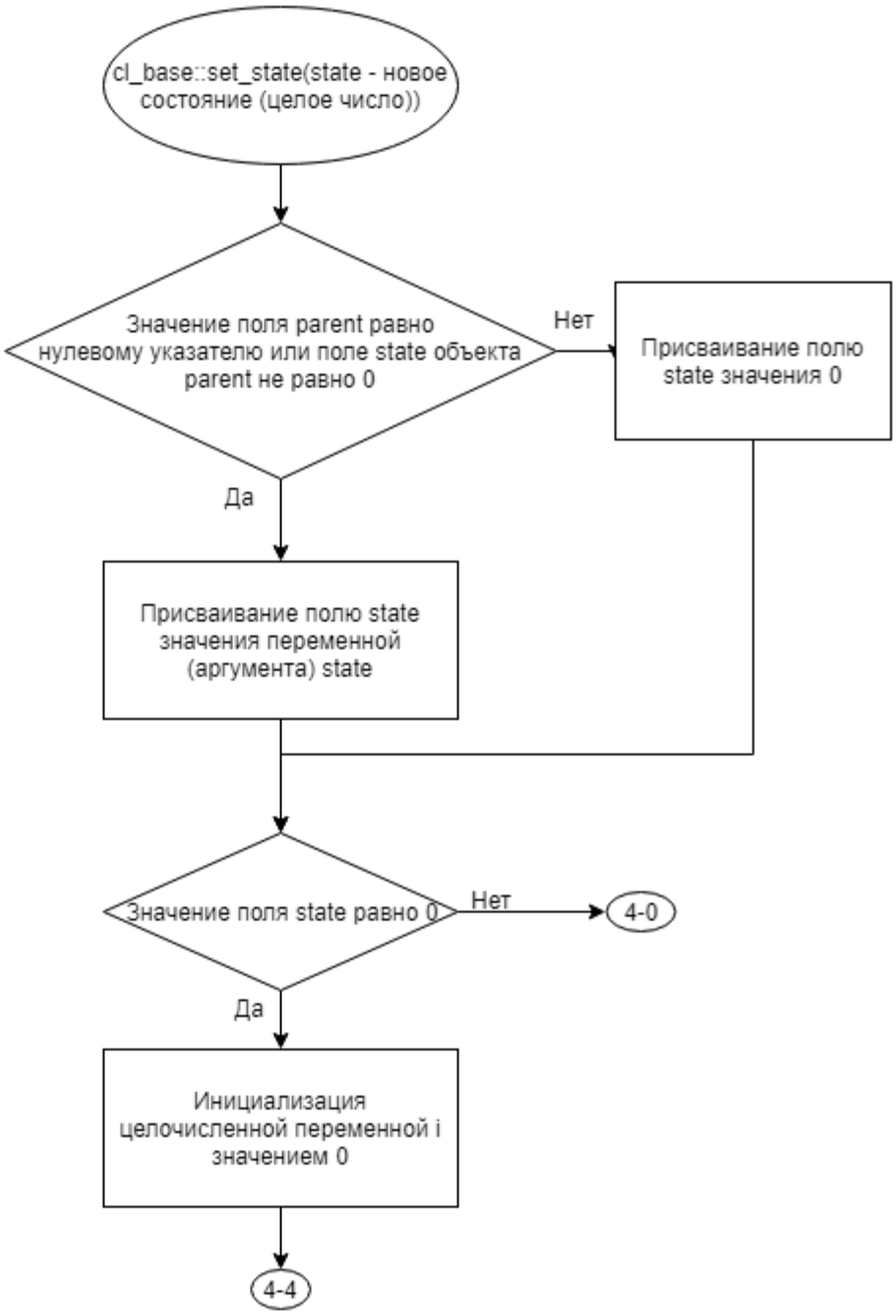

Рисунок 9 – Блок-схема алгоритма

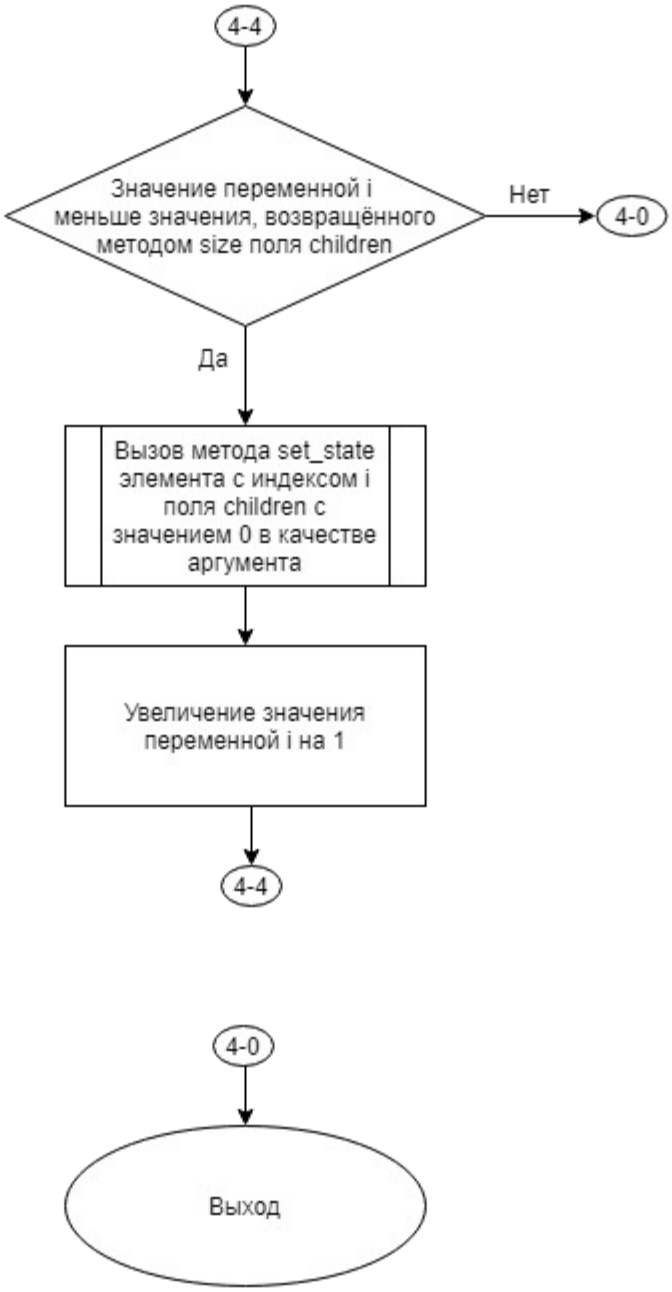

Рисунок 10 – Блок-схема алгоритма

Рисунок 11 – Блок-схема алгоритма

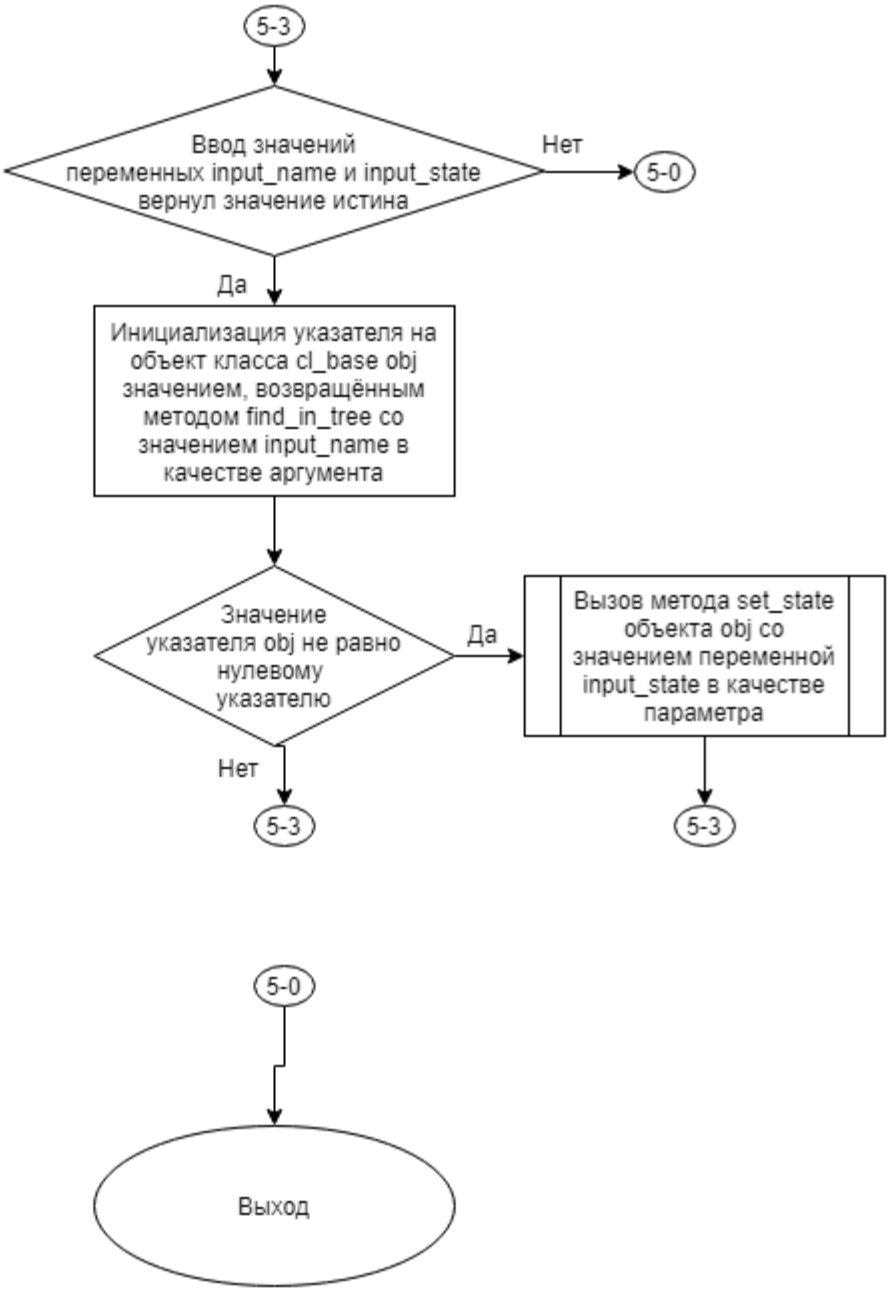

Рисунок 12 – Блок-схема алгоритма

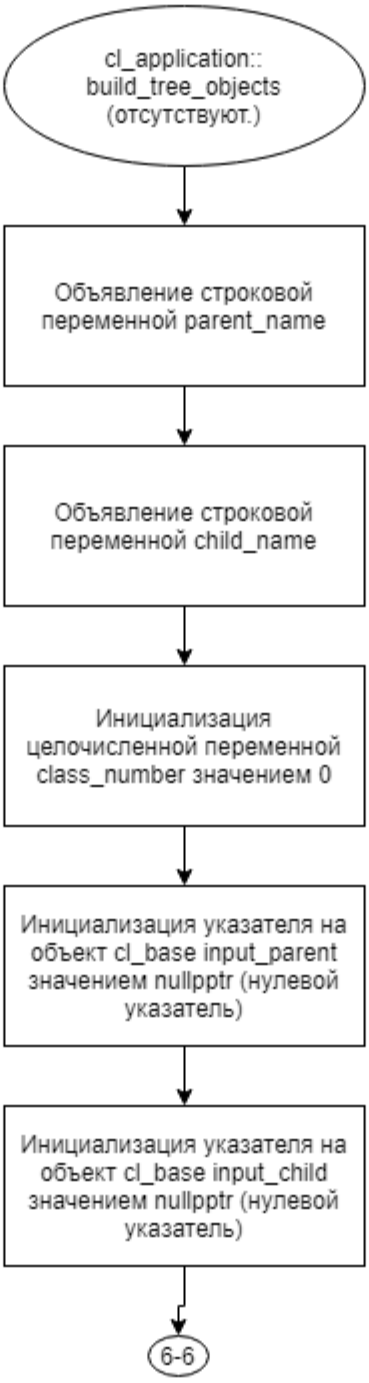

Рисунок 13 – Блок-схема алгоритма

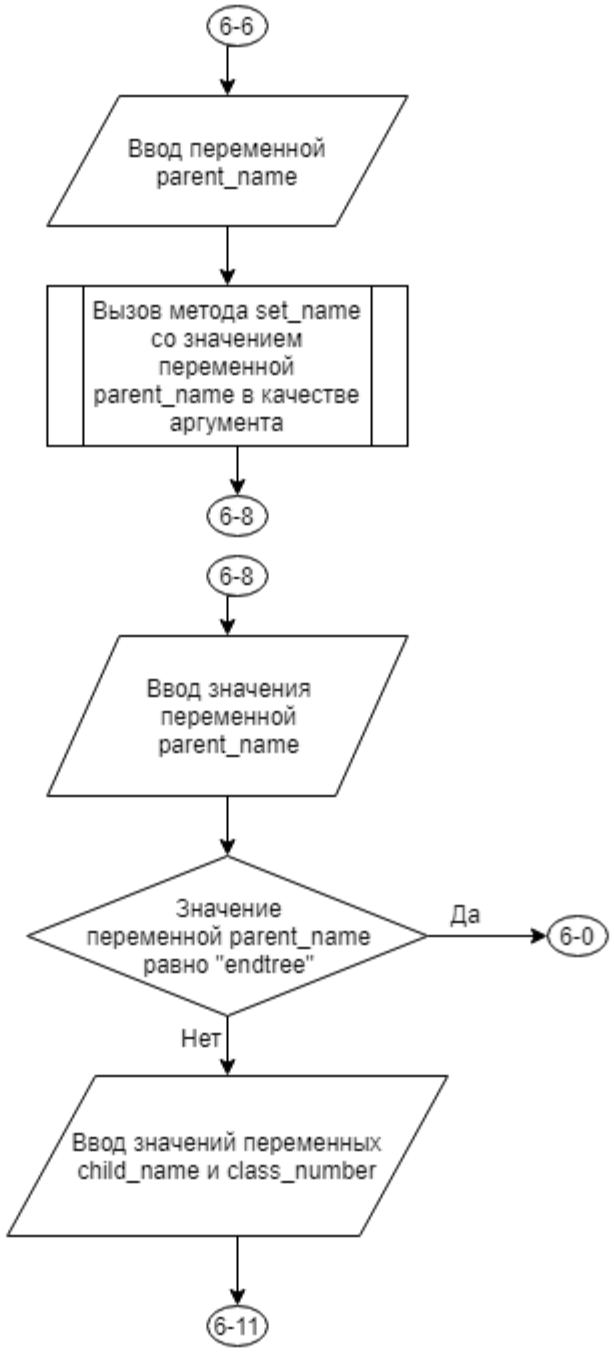

Рисунок 14 – Блок-схема алгоритма

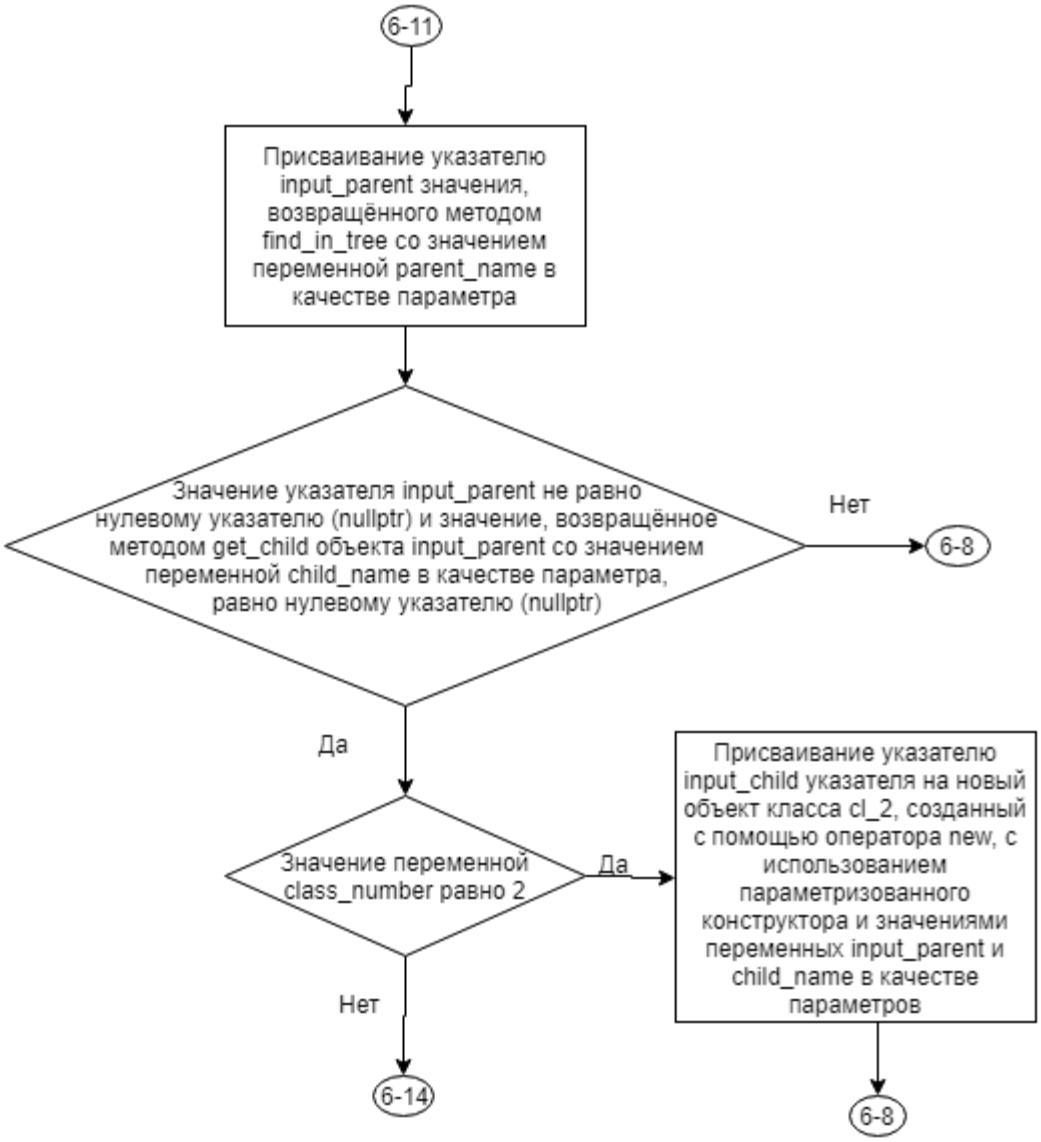

Рисунок 15 – Блок-схема алгоритма

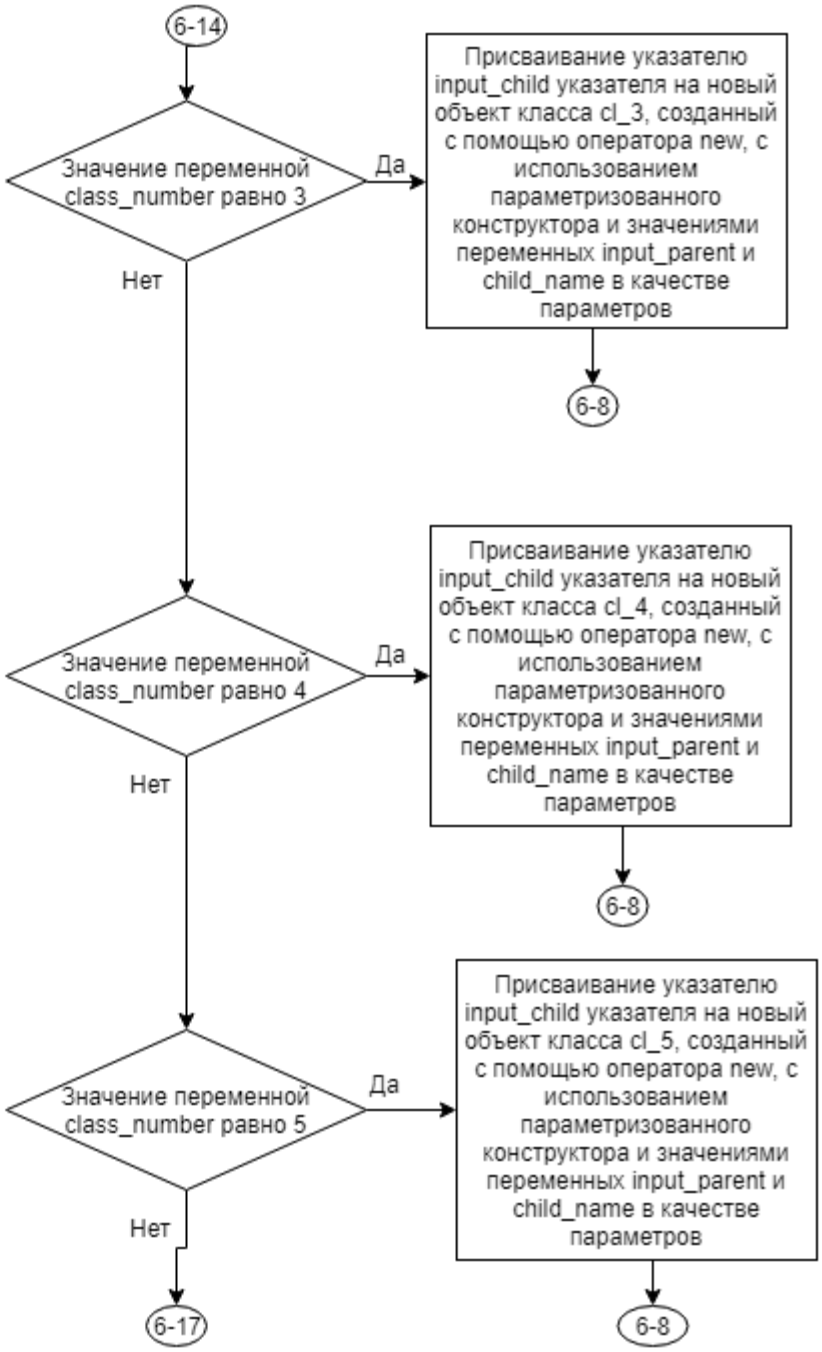

Рисунок 16 – Блок-схема алгоритма

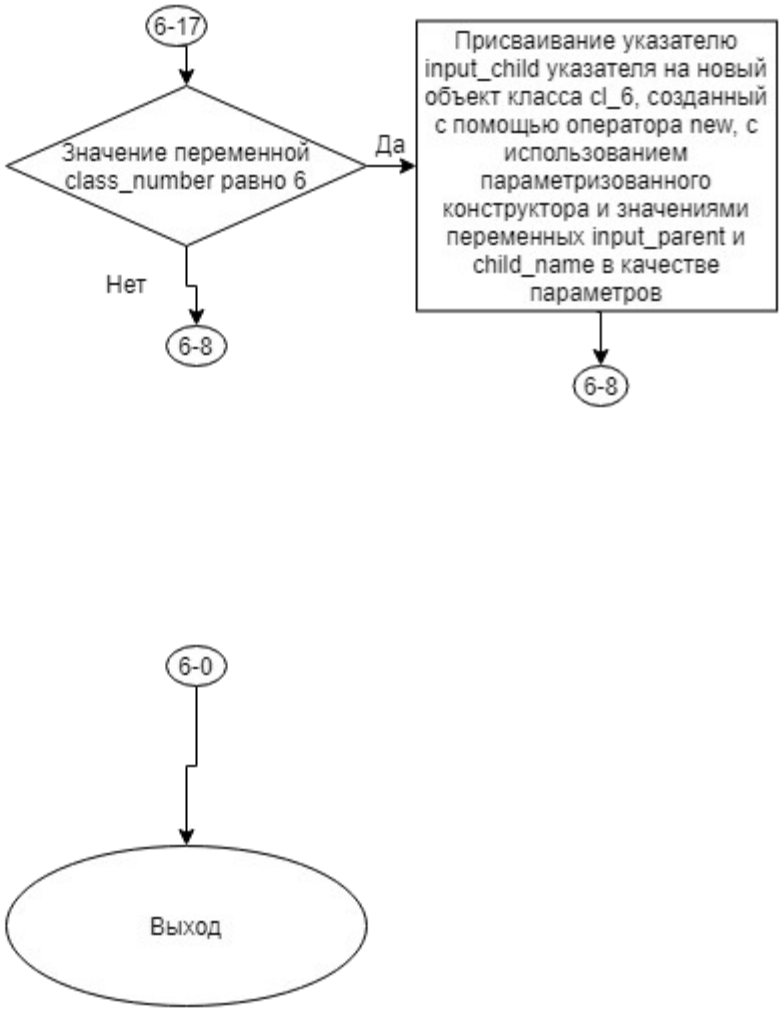

Рисунок 17 – Блок-схема алгоритма

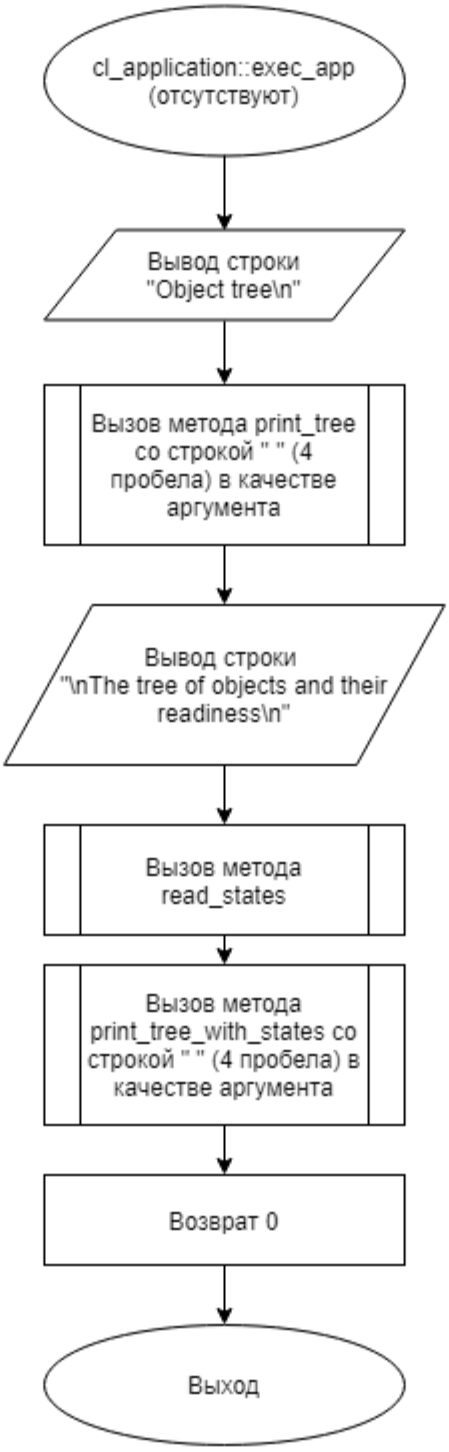

Рисунок 18 – Блок-схема алгоритма

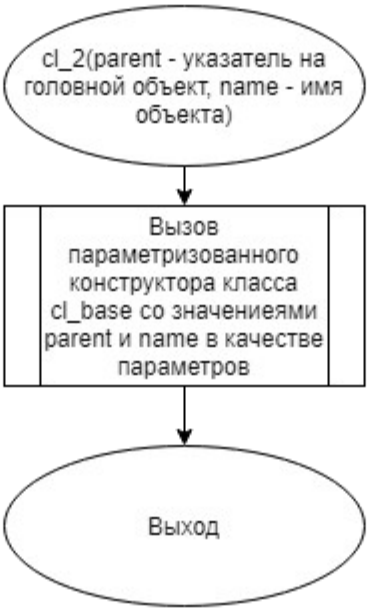

Рисунок 19 – Блок-схема алгоритма

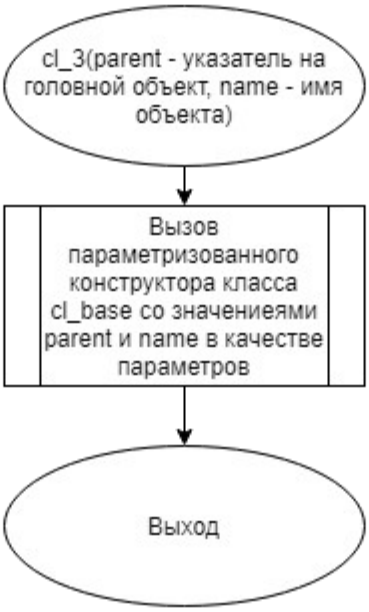

Рисунок 20 – Блок-схема алгоритма

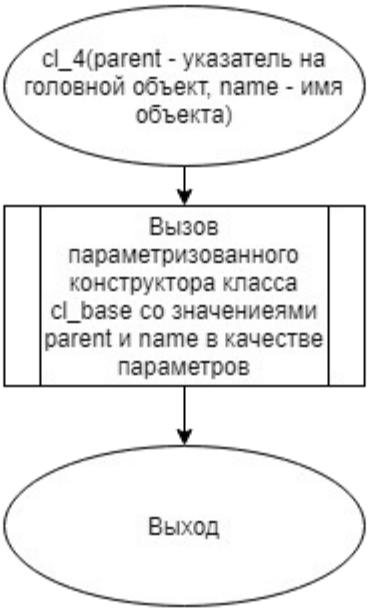

Рисунок 21 – Блок-схема алгоритма

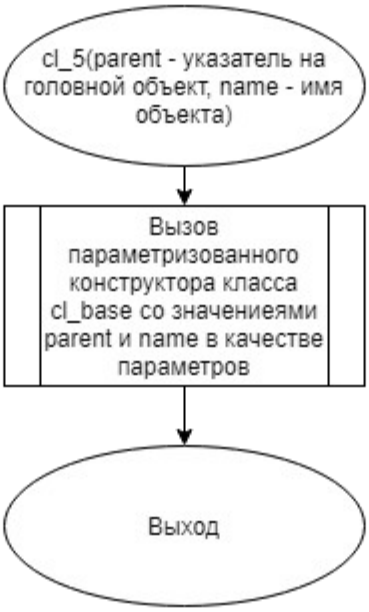

Рисунок 22 – Блок-схема алгоритма

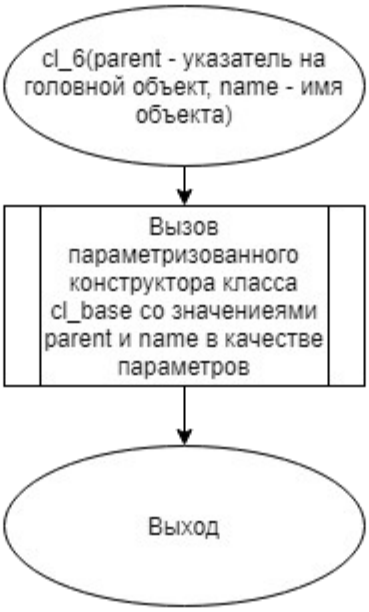

Рисунок 23 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base* parent, std::string name)
	: cl_base(parent, name) {}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H

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

class cl_2 : public cl_base
{
public:
	cl_2(cl_base* parent, std::string name);
};

#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* parent, std::string name)
	: cl_base(parent, name) {}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H

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

class cl_3 : public cl_base
{
public:
	cl_3(cl_base* parent, std::string name);
};

#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* parent, std::string name)
	: cl_base(parent, name) {}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H

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

class cl_4 : public cl_base
{
public:
	cl_4(cl_base* parent, std::string name);
};

#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* parent, std::string name)
      : cl_base(parent, name) {}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H

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

class cl_5 : public cl_base
{
public:
      cl_5(cl_base* parent, std::string name);
};

#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* parent, std::string name)
      : cl_base(parent, name) {}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H

```

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

class cl_6 : public cl_base
{
public:
	cl_6(cl_base* parent, std::string name);
};

#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – 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;
	int class_number = 0;
	cl_base* input_parent = nullptr;
	cl_base* input_child = this;
	cl_base* found_child = nullptr;

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

	while (true)
	{
		std::cin >> parent_name;
		if (parent_name == "endtree")
		{
			break;
		}

		std::cin >> child_name >> class_number;
		input_parent = input_child->find_in_branch(parent_name);
		found_child = find_in_tree(child_name);

		if (input_parent != nullptr && found_child == nullptr)
		{
			if (class_number == 2)
			{
				input_child = new cl_2(input_parent, child_name);
			}
			else if (class_number == 3)
			{
```

```
                        input_child = new cl_3(input_parent, child_name);
                }
                else if (class_number == 4)
                {
                        input_child = new cl_4(input_parent, child_name);
                }
                else if (class_number == 5)
                {
                        input_child = new cl_5(input_parent, child_name);
                }
                else if (class_number == 6)
                {
                        input_child = new cl_6(input_parent, child_name);
                }
        }
    }
}

// Метод запуска приложения
int cl_application::exec_app()
{
    std::cout << "Object tree\n";
    print_tree("    ");
    std::cout << "\nThe tree of objects and their readiness\n";
    read_states();
    print_tree_with_state("    ");
    return 0;
}

void cl_application::read_states()
{
    std::string input_name;
    int input_state = 0;

    while (std::cin >> input_name >> input_state)
    {
        cl_base* obj = find_in_tree(input_name);
        if (obj != nullptr)
        {
            obj->set_state(input_state);
        }
    }
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H

#include "cl_base.h"
#include "cl_2.h"
```

```
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.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();

	// Метод для считывания имён объектов и их готовностей
	void read_states();
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – 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(std::string prefix)
{
	if (parent == nullptr)
	{
		// Вывод имени текущего объекта
		std::cout << name;
	}

	// Вывод имён всех подчинённых объектов
	for (int i = 0; i < children.size(); i++)
	{
		std::cout << '\n' << prefix << children[i]->name;
		children[i]->print_tree(prefix + "    ");
	}
}

// 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;
}

cl_base* cl_base::find_in_tree(std::string wanted_name)
{
      if (name == wanted_name)
      {
            return this;
      }

      for (int i = 0; i < children.size(); i++)
      {
            cl_base* found = children[i]->find_in_tree(wanted_name);
            if (found != nullptr)
            {
                  return found;
            }
      }

      return nullptr;
}

void cl_base::print_tree_with_state(std::string prefix)
{
      if (parent == nullptr)
      {
            std::cout << name << (state == 0 ? " is not ready" : " is ready");
      }

      for (int i = 0; i < children.size(); i++)
      {
            std::cout << "\n" << prefix << children[i]->get_name()
                  << (children[i]->get_state() == 0 ? " is not ready" : " is
ready");
            children[i]->print_tree_with_state(prefix + "    ");
      }
}

int cl_base::get_state()
{
      return state;
}

void cl_base::set_state(int state)
{
      if (parent == nullptr || parent->state != 0)
      {
            this->state = state;
      }
      else
```

```
	{
		this->state = 0;
	}

	if (state == 0)
	{
		for (int i = 0; i < children.size(); i++)
		{
			children[i]->set_state(0);
		}
	}
}

cl_base* cl_base::get_root()
{
	if (parent == nullptr)
	{
		return this;
	}
	else
	{
		return parent->get_root();
	}
}

cl_base* cl_base::find_in_branch(std::string wanted_name)
{
	int amount = get_root()->count_in_tree(wanted_name);
	if (amount == 1)
	{
		return get_root()->find_in_tree(wanted_name);
	}

	return nullptr;
}

int cl_base::count_in_tree(std::string wanted_name)
{
	int res = 0;
	if (name == wanted_name)
	{
		res++;
	}

	for (int i = 0; i < children.size(); i++)
	{
		res += children[i]->count_in_tree(wanted_name);
	}

	return res;
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H

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

class cl_base
{
private:

	int state;

	// 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(std::string prefix);

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

	// Метод поиска объекта по заданному имени в дереве иерархии объектов
	cl_base* find_in_tree(std::string wanted_name);

	// Метод вывода наименований объектов в дереве иерархии слева
	// направо и сверху вниз вместе с их готовностью
	void print_tree_with_state(std::string prefix);
```

```
	// Метод получения состояния объекта
	int get_state();

	// Метод получения состояния объекта
	void set_state(int state);

	cl_base* get_root();

	cl_base* find_in_branch(std::string wanted_name);

	int count_in_tree(std::string wanted_name);
};

#endif
```

5.15 Файл main.cpp

Листинг 15 – 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 19.

Таблица 19 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
	Object tree
	app_root
	    object_01
	        object_07
	    object_02
	        object_04
	        object_05
	The tree of objects and their readiness
	app_root is ready
	    object_01 is ready
	          object_07 is not ready
	    object_02 is ready
	        object_04 is ready
	          object_05 is not ready
		Object tree
		app_root
		    object_01
		        object_07
		    object_02
		        object_04
		        object_05
		The tree of objects and their readiness
		app_root is ready
		    object_01 is ready
		          object_07 is not ready
		    object_02 is ready
		        object_04 is ready
		          object_05 is not ready

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).