

Здесь будет титульник, листай ниже

# СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ.....	6
1.1 Описание входных данных.....	8
1.2 Описание выходных данных.....	9
2 МЕТОД РЕШЕНИЯ.....	10
3 ОПИСАНИЕ АЛГОРИТМОВ.....	13
3.1 Алгоритм конструктора класса cl_base.....	13
3.2 Алгоритм деструктора класса cl_base.....	13
3.3 Алгоритм метода change_name класса cl_base.....	14
3.4 Алгоритм метода get_name класса cl_base.....	15
3.5 Алгоритм метода get_p_head класса cl_base.....	15
3.6 Алгоритм метода show_tree класса cl_base.....	16
3.7 Алгоритм метода get_subordinate_object класса cl_base.....	16
3.8 Алгоритм конструктора класса cl_1.....	17
3.9 Алгоритм конструктора класса cl_application.....	18
3.10 Алгоритм метода build_tree_objects класса cl_application.....	18
3.11 Алгоритм метода exes_app класса cl_application.....	19
3.12 Алгоритм функции main.....	20
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....	21
5 КОД ПРОГРАММЫ.....	30
5.1 Файл cl_1.cpp.....	30
5.2 Файл cl_1.h.....	30
5.3 Файл cl_application.cpp.....	30
5.4 Файл cl_application.h.....	31
5.5 Файл cl_base.cpp.....	32
5.6 Файл cl_base.h.....	33
5.7 Файл main.cpp.....	33

6 ТЕСТИРОВАНИЕ.....	35
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	36

# 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:
  - о наименование объекта (строкового типа);
  - о указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно nullptr);
  - о динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.
- функционал:
  - о 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);
  - о метод редактирования имени объекта. Один параметр строкового типа, содержит новое наименование объекта. Если нет дуближа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
  - о метод получения имени объекта;

- о метод получения указателя на головной объект текущего объекта;
- о метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- о 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 приложения. Класс объекта приложения наследуется от базового класса. Объект приложения 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 используется:

- объект `ob_cl_application` класса `cl_application` предназначен для объект приложения, предназначен для построения дерева и запуска приложения;
- функция `main` для основной алгоритм программы;
- `cin/cout` - операторы потока ввода/вывода;
- `if` - условный оператор;
- `for` - цикл со счетчиком;
- `while` - цикл с условием ;
- объекты класса `cl_1` количество которых определено самим пользователем.

Класс `cl_base`:

- свойства/поля:
  - поле наименование объекта:
    - наименование — `s_object_name`;
    - тип — `string`;
    - модификатор доступа — `private`;
  - поле указатель на головной объект для текущего объекта:
    - наименование — `p_head_object`;
    - тип — `cl_base*`;
    - модификатор доступа — `private`;
  - поле динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии:
    - наименование — `subordinate_objects`;
    - тип — `vector <cl_base* >`;
    - модификатор доступа — `private`;



- функционал:
  - о метод `cl_base` — параметризированный конструктор класса;
  - о метод `~cl_base` — деструктор класса;
  - о метод `change_name` — метод редактирования имени объекта;
  - о метод `get_name` — метод получения имени объекта;
  - о метод `get_p_head` — метод получения указателя на головной объект текущего объекта;
  - о метод `show_tree` — метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
  - о метод `get_subordinate_object` — метод получения указателя на непосредственно подчиненный объект по его имени.

Класс `cl_1`:

- функционал:
  - о метод `cl_1` — параметризированный конструктор класса.

Класс `cl_application`:

- функционал:
  - о метод `cl_application` — параметризированный конструктор класса;
  - о метод `build_tree_objects` — метод построения исходного дерева иерархии объектов;
  - о метод `exes_app` — метод запуска приложения.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс	
		cl_1	public		2
		cl_applicatio n	public		3

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
2	cl_1			Класс наследуемый от класса cl_base	
3	cl_application			Класс объекта приложения	

## 3 ОПИСАНИЕ АЛГОРИТМОВ

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

### 3.1 Алгоритм конструктора класса `cl_base`

Функционал: параметризованный конструктор класса.

Параметры: `cl_base * p_head_object` - указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии `string s_object_name` - наименование создаваемого объекта, по умолчанию подается "Base object" .

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса `cl_base`

№	Предикат	Действия	№ перехода
1		Полю <code>p_head_object</code> этого объекта присваивается параметр <code>p_head_object</code>	2
2		Полю <code>s_object_name</code> этого объекта присваивается параметр <code>s_object_name</code>	3
3	<code>p_head_object</code> не нулевой	В вектор указателей подчиненных объектов головного объекта добавляется указатель на этот объект	∅
			∅

### 3.2 Алгоритм деструктора класса `cl_base`

Функционал: деструктор класса.

Параметры: нет.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса *cl\_base*

№	Предикат	Действия	№ перехода
1		Инициализация <i>i</i> значением 0	2
2	<i>i</i> < длины subordinate_objects	Удаляем объект по <i>i</i> -ому указателю вектора указателей subordinate_objects	3
			∅
3		<i>i</i> = <i>i</i> + 1	2

### 3.3 Алгоритм метода *change\_name* класса *cl\_base*

Функционал: метод редактирования имени объекта.

Параметры: string new\_name - новое наименование объекта.

Возвращаемое значение: bool - Если нет дуближа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь».

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода *change\_name* класса *cl\_base*

№	Предикат	Действия	№ перехода
1	<i>p_head_object</i> этого объекта не нулевой	Инициализация <i>i</i> значением 0	2
			5
2	<i>i</i> < длины subordinate_objects объекта по указателю <i>p_head_object</i>		3
			5
3	Имя объекта по <i>i</i> -ому указателю вектора указателей subordinate_objects объекта	Возвращаем «ложь»	∅

№	Предикат	Действия	№ перехода
	по указателю p_head_object равно new_string		
			4
4		i = i + 1	2
5		полю s_object_name этого объекта присваивается значение new_name	6
6		Возвращаем «истину»	∅

### 3.4 Алгоритм метода get\_name класса cl\_base

Функционал: метод получения имени объекта.

Параметры: нет.

Возвращаемое значение: string - имя этого объекта.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get\_name класса cl\_base

№	Предикат	Действия	№ перехода
1		Возвращаем поле имя объекта s_object_name	∅

### 3.5 Алгоритм метода get\_p\_head класса cl\_base

Функционал: метод получения указателя на головной объект текущего объекта.

Параметры: нет.

Возвращаемое значение: cl\_base\* - указатель на головной объект текущего объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода *get\_p\_head* класса *cl\_base*

№	Предикат	Действия	№ перехода
1		Возвращаем указатель на головной объект текущего объекта p_head_object	Ø

### 3.6 Алгоритм метода *show\_tree* класса *cl\_base*

Функционал: метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода *show\_tree* класса *cl\_base*

№	Предикат	Действия	№ перехода
1	длина subordinate_objects не равна 0	Вывод символа переноса строки и возвращаемого значения метода get_name этого объекта	2
			Ø
2		Инициализация i значением 0	3
3	i < длины subordinate_objects	Вывод " " и возвращаемого значения метода get_name объекта по i-ому указателю вектора указателей subordinate_objects	4
			Ø
4		Вызов метода show_tree для объекта по i-ому указателю вектора указателей subordinate_objects	5
5		i = i + 1	3

### 3.7 Алгоритм метода *get\_subordinate\_object* класса *cl\_base*

Функционал: метод получения указателя на непосредственно подчиненный

объект по его имени.

Параметры: string search\_name - наименование искомого подчиненного объекта.

Возвращаемое значение: cl\_bas\* - указатель на непосредственно подчиненный объект по его имени.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get\_subordinate\_object класса cl\_base

№	Предикат	Действия	№ перехода
1		Инициализация i значением 0	2
2	i < длины subordinate_objects		3
			5
3	поле s_object_name объекта по i-ому указателю вектора subordinate_objects равно search_name	Возвращаем объект по i-ому указателю вектора указателей subordinate_objects	∅
			4
4		i = i + 1	2
5		Возвращаем нулевой указатель	∅

### 3.8 Алгоритм конструктора класса cl\_1

Функционал: параметризованный конструктор класса.

Параметры: cl\_base \* p\_head\_object - указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии string s\_object\_name - наименование создаваемого объекта.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса *cl\_1*

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса <i>cl_base</i> с параметрами <i>p_head_object</i> и <i>s_object_name</i>	∅

### 3.9 Алгоритм конструктора класса *cl\_application*

Функционал: параметризованный конструктор класса.

Параметры: *cl\_base \* new\_root* - указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии .

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса *cl\_application*

№	Предикат	Действия	№ перехода
1		Вызов параметризованного конструктора класса <i>cl_base</i> с параметром <i>new_root</i>	∅

### 3.10 Алгоритм метода *build\_tree\_objects* класса *cl\_application*

Функционал: метод построения исходного дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: *void*.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода *build\_tree\_objects* класса *cl\_application*

№	Предикат	Действия	№ перехода
1		Объявление строк <i>s_head</i> и <i>s_sub</i>	2
2		Объявление указателя <i>p_head</i> на объект класса <i>cl_base</i>	3
3		Инициализируем указатель <i>p_sub</i> на объект класса	4



№	Предикат	Действия	№ перехода
		cl_base нулевым	
4		Ввод с клавиатуры значения s_head	5
5		Вызов метода change_name этого объекта с параметром s_head	6
6		Присваиваем объекту по указателю p_head этот объект	7
7		Ввод с клавиатуры значения s_head и s_sub	8
8	s_head равно s_sub		∅
			9
9	p_sub не нулевой и s_head равно возвращаемому значению метода get_name объекта по указателю p_sub	Присваиваем p_head значение p_sub	10
			10
10	Возвращаемое значение метода get_subordinate_object с параметром s_sub объекта по указателю p_head - нулевой, и s_head равно возвращаемому значению метода get_name объекта по указателю p_head	Присваиваем объекту по указателю p_sub новый объект класса cl_1 с параметрами p_head и s_sub	7
			7

### 3.11 Алгоритм метода exes\_app класса cl\_application

Функционал: метод запуска приложения.

Параметры: нет.

Возвращаемое значение: int - целое индикатор корректности выполнения

программы.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода *exes\_app* класса *cl\_application*

№	Предикат	Действия	№ перехода
1		Вывод значения возвращаемого значения метода <i>get_name</i> этого объекта	2
2		Вызов метода <i>show_tree</i> этого объекта	3
3		Возвращаем 0	Ø

### 3.12 Алгоритм функции *main*

Функционал: Основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: *int* - целое индификатор корректности выполнения программы.

Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции *main*

№	Предикат	Действия	№ перехода
1		Создаем объект <i>ob_cl_application</i> класса <i>cl_application</i> в качестве параметра подается нулевой указатель	2
2		Вызываем метод <i>build_tree_objects</i> объекта <i>ob_cl_application</i>	3
3		Вызываем метод <i>exes_app</i> объекта <i>ob_cl_application</i>	Ø

## 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-9.

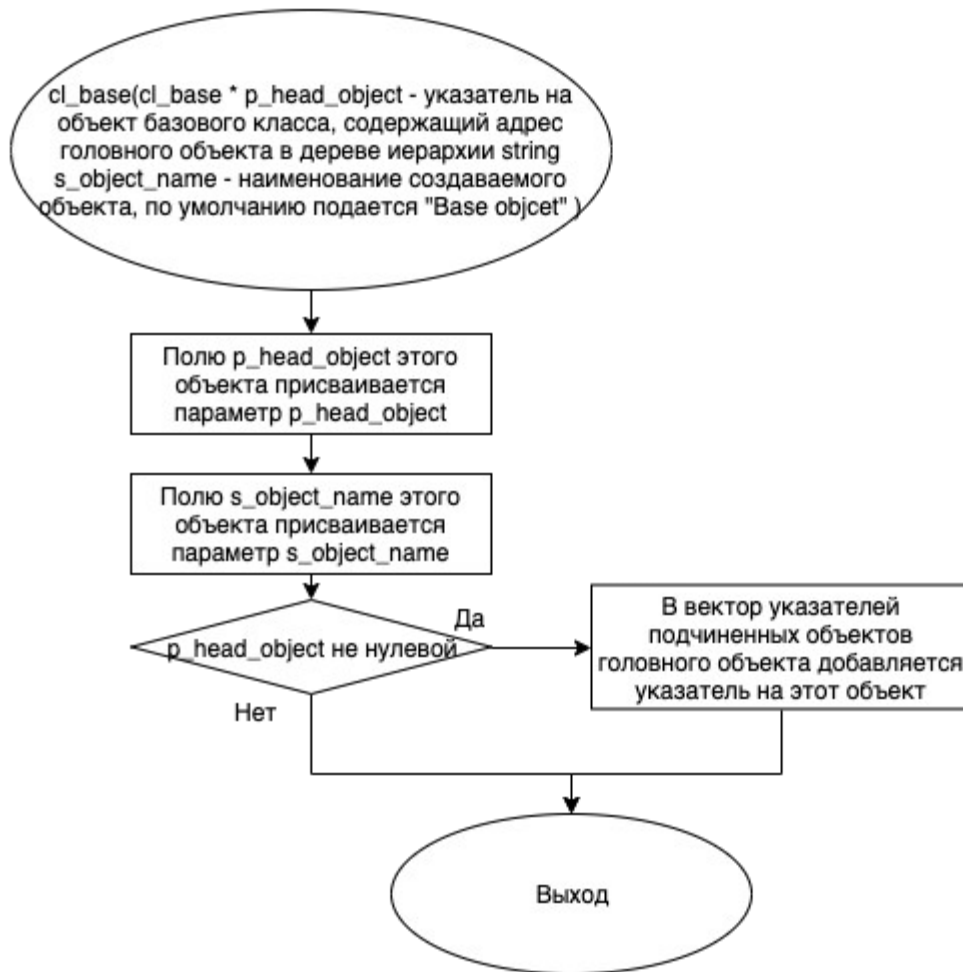


Рисунок 1 – Блок-схема алгоритма

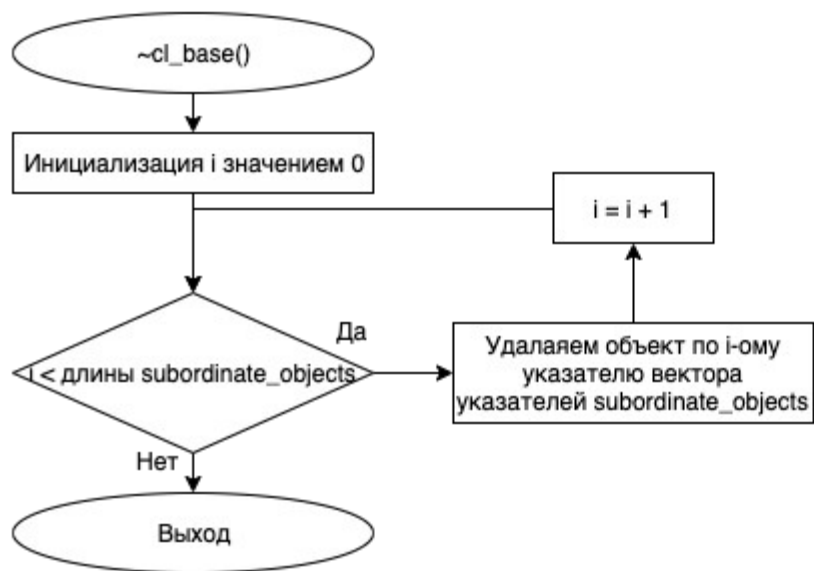


Рисунок 2 – Блок-схема алгоритма

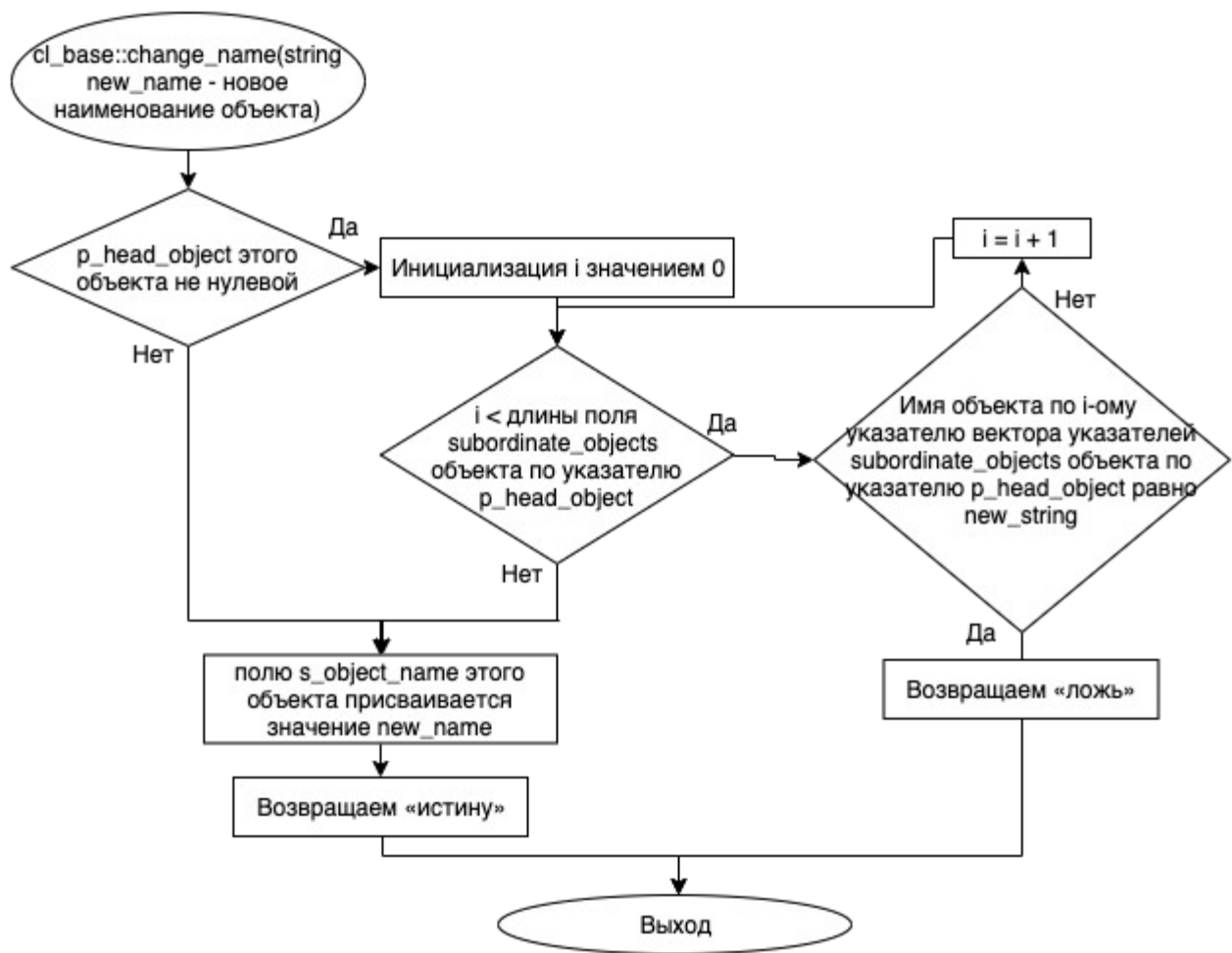


Рисунок 3 – Блок-схема алгоритма



**Рисунок 4 – Блок-схема алгоритма**

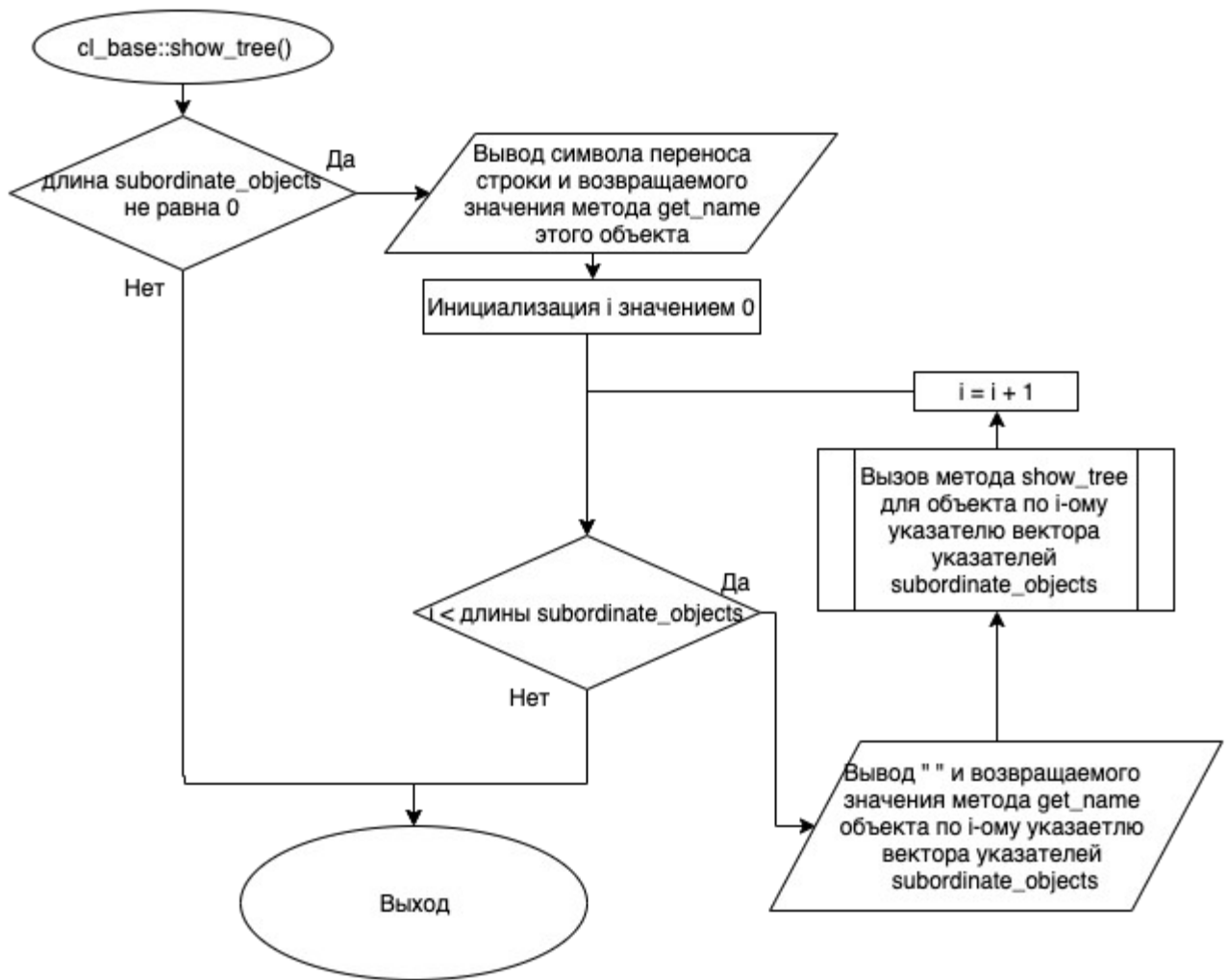


Рисунок 5 – Блок-схема алгоритма

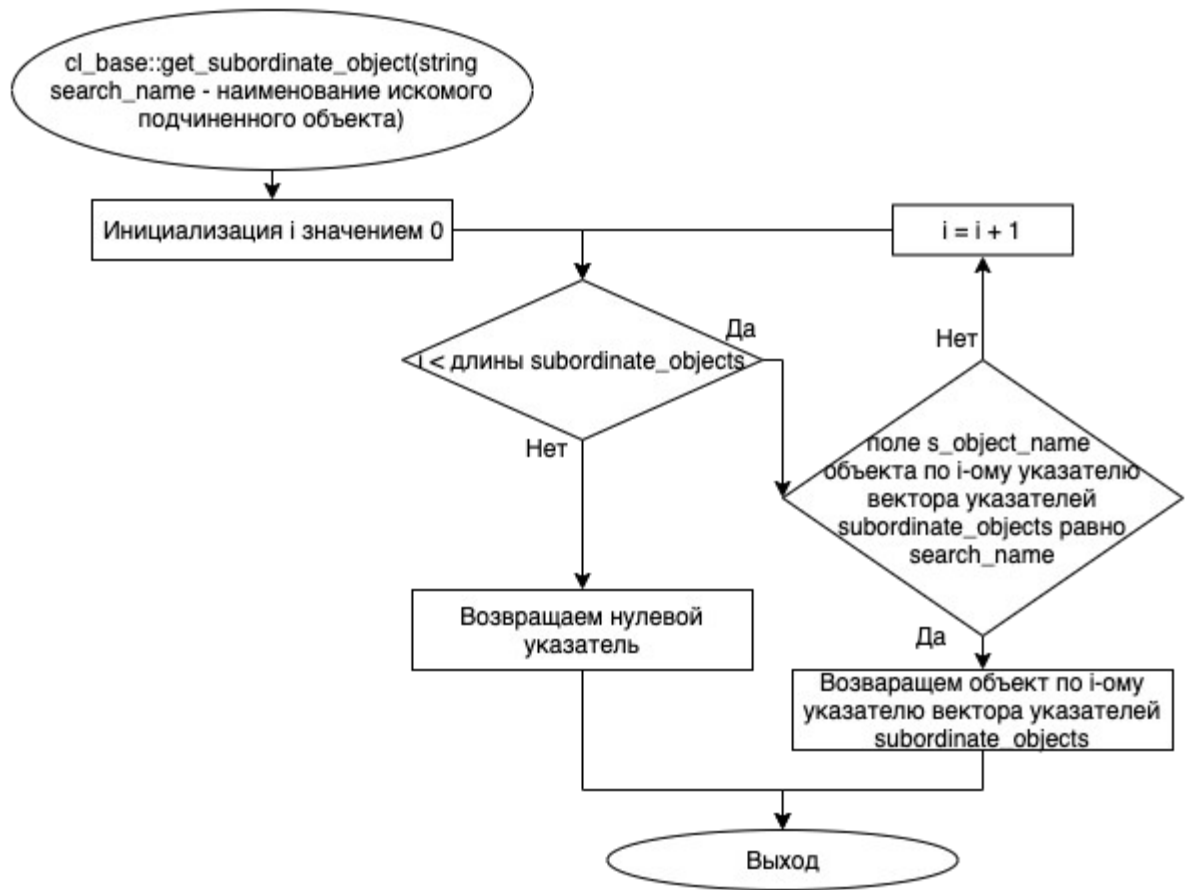


Рисунок 6 – Блок-схема алгоритма



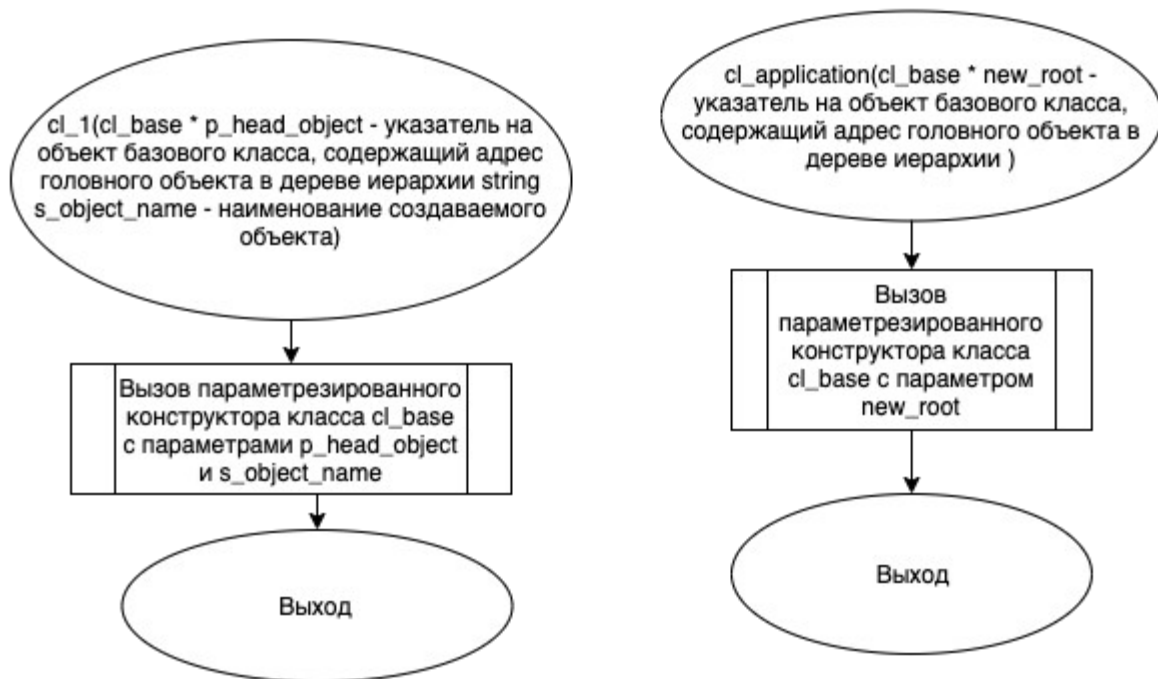


Рисунок 7 – Блок-схема алгоритма

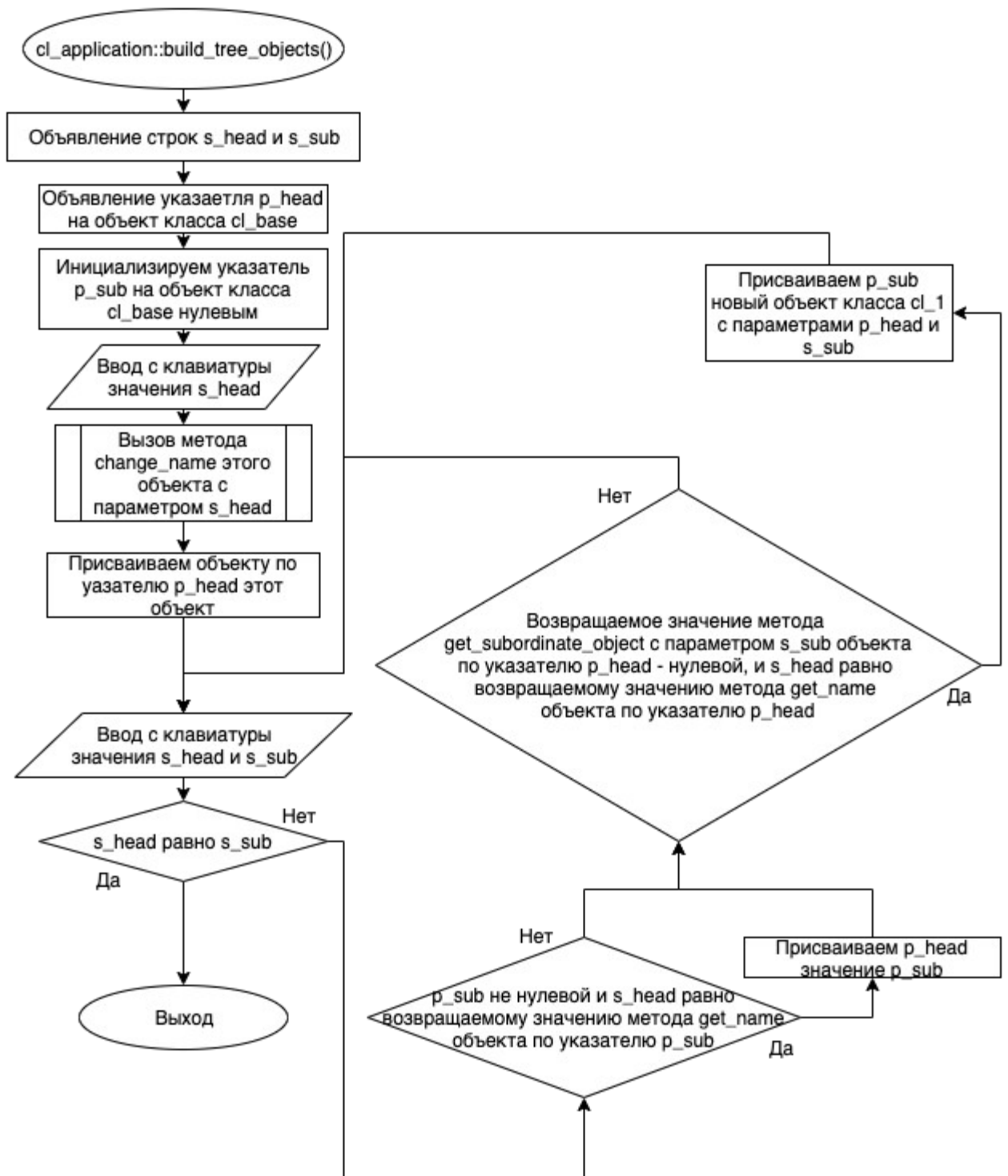


Рисунок 8 – Блок-схема алгоритма

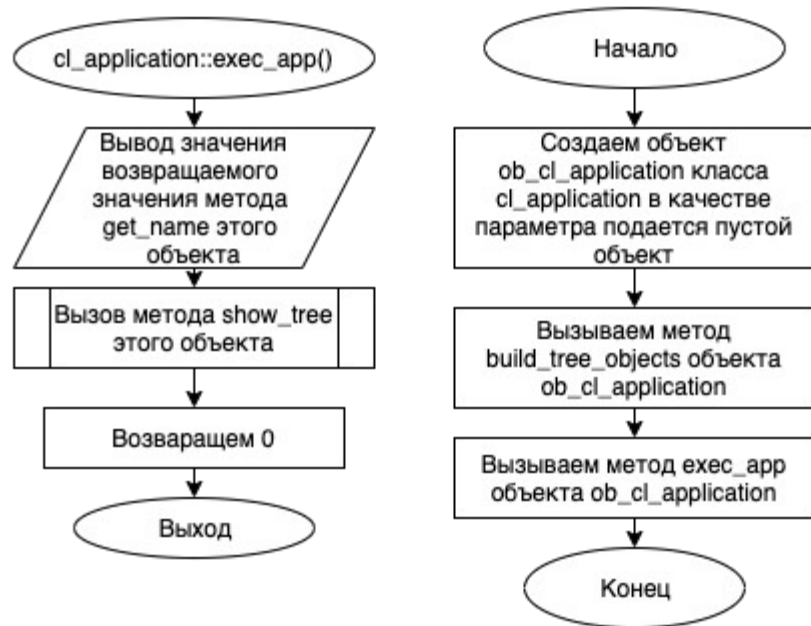


Рисунок 9 – Блок-схема алгоритма

## 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

### 5.1 Файл cl\_1.cpp

*Листинг 1 – cl\_1.cpp*

```
#include "cl_1.h"

cl_1::cl_1(cl_base* p_head_object, string s_object_name):cl_base(p_head_object, s_object_name){}
```

### 5.2 Файл cl\_1.h

*Листинг 2 – cl\_1.h*

```
#ifndef __CL_1__H
#define __CL_1__H
#include "cl_base.h"

class cl_1 : public cl_base {
public:
    cl_1(cl_base* p_head_object, string s_object_name);
};

#endif
```

### 5.3 Файл cl\_application.cpp

*Листинг 3 – cl\_application.cpp*

```
#include "cl_application.h"

cl_application::cl_application(cl_base * new_root) : cl_base(new_root){}
```

```

void cl_application::build_tree_objects() {
    string s_head, s_sub;
    cl_base* p_head;
    cl_base* p_sub = nullptr;
    cin >> s_head;
    this -> change_name(s_head);
    p_head = this;

    while(true) {
        cin >> s_head >> s_sub;
        if (s_head == s_sub) {
            break;
        }
        if (p_sub != nullptr && s_head == p_sub -> get_name()) {
            p_head = p_sub;
        }
        if (p_head -> get_subordinate_object(s_sub) == nullptr && s_head ==
p_head -> get_name()) {
            p_sub = new cl_1(p_head, s_sub);
        }
    }
}

int cl_application::exec_app() {
    cout << this -> get_name();
    this -> show_tree();
    return 0;
}

```

## 5.4 Файл cl\_application.h

*Листинг 4 – cl\_application.h*

```

#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
#include "cl_1.h"

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

#endif

```

## 5.5 Файл cl\_base.cpp

Листинг 5 – cl\_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base * p_head_object, string s_object_name) {
    this -> p_head_object = p_head_object;
    this -> s_object_name = s_object_name;
    if (p_head_object) {
        p_head_object -> subordinate_objects.push_back(this);
    }
}

bool cl_base::change_name(string new_name) {
    if (this -> p_head_object) {
        for (int i = 0; i < p_head_object -> subordinate_objects.size(); i++) {
            if (p_head_object -> subordinate_objects[i] -> get_name() ==
new_name) {
                return false;
            }
        }
    }
    this -> s_object_name = new_name;
    return true;
}

string cl_base::get_name() {
    return s_object_name;
}

cl_base * cl_base::get_p_head() {
    return p_head_object;
}

void cl_base::show_tree() {
    if (subordinate_objects.size() != 0){
        cout << endl << this -> get_name();
        for (int i = 0; i < subordinate_objects.size(); i++) {
            cout << " " << subordinate_objects[i] -> get_name();
            subordinate_objects[i] -> show_tree();
        }
    }
}

cl_base * cl_base::get_subordinate_object(string search_name) {
    for (int i = 0; i < subordinate_objects.size(); i++) {
        if (subordinate_objects[i] -> s_object_name == search_name) {
            return subordinate_objects[i];
        }
    }
    return nullptr;
}

cl_base::~cl_base() {
```

```

        for (int i = 0; i < subordinate_objects.size(); i++) {
            delete subordinate_objects[i];
        }
    }
}

```

## 5.6 Файл cl\_base.h

Листинг 6 – cl\_base.h

```

#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <vector>

using namespace std;

class cl_base {
public:
    cl_base(cl_base * p_head_object, string s_object_name = "Base object");
    bool change_name(string new_name);
    string get_name();
    cl_base * get_p_head();
    void show_tree();
    cl_base * get_subordinate_object(string search_name);
    ~cl_base();
private:
    string s_object_name;
    cl_base * p_head_object;
    vector <cl_base * > subordinate_objects;
};

#endif

```

## 5.7 Файл main.cpp

Листинг 7 – main.cpp

```

#include <stdlib.h>
#include <stdio.h>
#include "cl_base.h"
#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 14.

Таблица 14 – 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. 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-avvora.ru/student/files/methodichescoe\\_posobie\\_dlya\\_laboratornyh\\_rabot\\_3.pdf](https://mirea.aco-avvora.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-avvora.ru/student/files/Prilozheniye\\_k\\_methodichke.pdf](https://mirea.aco-avvora.ru/student/files/Prilozheniye_k_methodichke.pdf) (дата обращения 05.05.2021).
4. Шилдт Г. С++: 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).