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

Объекты потока ввода/вывода cin, cout;

Оператор функция new;

Оператор функция delete;

Объект ob_application классса application;

Объекты элемента дерева класса cl_node, количество которых определяется вводом пользователя;

Класс cl_base:

- Свойства/поля:
  - Поле, отвечающее за наименование объекта:
    - Наименование - name;
    - Тип - string;
    - Модификатор доступа - private;
  - Поле, отвечающее за доступ к головному объекту текущего объекта:
    - Наименование - parent;
    - Тип - указатель на объект класса cl_base;
    - Модификатор доступа - private;
  - Поле, отвечающее за доступ к подчиненным объектам текущего объекта:
    - Наименование - children;
    - Тип - контейнер класса vector с указателями на объекты класса cl_base;
    - Модификатор доступа - private;
- Функционал:
  - Параметризированный конструктор cl_base с параметром указателя на головной объект в дереве иерархии и наименованием объекта -

принимающим значение по умолчанию "Object root";

- Деструктор ~cl_base используется для освобождения памяти выделенной для иерархии объектов;
- Метод setName испольуется для установки имени текущего объекта;
- Метод get_name используется для получения имени текущего объекта;
- Метод get_parent используется для получения указателя на головной объект текущего объекта;
- Метод get_child_by_name используется для получения указателя на объект из дерева иерархии объектов;
- Метод print_names вывод наименования объектов в дереве иерархии слева направо и сверху вниз;

Класс application:

- Свойства/поля:
  - Поля, унаследованные от класса cl_base;
- Функционал:
  - Методы, унаследованные от класса cl_base;
  - Параметризированный конструктор application, с параметром указателя на головной объект в дереве иерархии
  - Метод build_tree_objects используется для построения дерева иерархии объектов;
  - Метод exec_app используется для запуска системы;

Класс cl_node:

- Свойства/поля:
  - Поля, унаследованные от класса cl_base;
- Функционал:
  - Методы, унаследованные от класса cl_base;
  - параметризированный конструктор cl_node с параметром указателя на

головной объект в дереве иерархии;

*Таблица 1 – Иерархия наследования классов*

| № | Имя класса | Классы-наследники | Модификаторы доступа при наследовании | Описание | Номер | Комментарий |
|---|---|---|---|---|---|---|
| 1 | cl_base | | | Базовый класс в иерархии классов. Содержит основные свойства и методы | | |
| | | application | public | | 2 | |
| | | cl_node | public | | 3 | |
| 2 | application | | | Класс корневого объекта (приложения) | | |
| 3 | cl_node | | | Класс объекта дерева | | |

# 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: Основная программа.

Параметры: .

Возвращаемое значение: int - код возврата.

Алгоритм функции представлен в таблице 2.

*Таблица 2 – Алгоритм функции main*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Создание объекта ob_application класса application с использованием параметрезированного конструктора и передачей в него в качестве параметра пустого указателя | 2 |
| 2 | | Вызов метода buildTree объекта ob_application | 3 |
| 3 | | Возвращение результата работы метода exec_app() для объекта ob_application | ∅ |

## 3.2 Алгоритм конструктора класса cl_base

Функционал: Вызывается при создании объекта, инициализирует значения приватных полей.

Параметры: Node* ptr_parent, string _name.

Алгоритм конструктора представлен в таблице 3.

*Таблица 3 – Алгоритм конструктора класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | parent != nullptr | Запись данного объекта в конец вектора children, принадлежащего объекту parent | ∅ |
| | | | ∅ |

## 3.3 Алгоритм деструктора класса cl_base

Функционал: Вызывается при уничтожении объекта и очищает память.

Параметры: .

Алгоритм деструктора представлен в таблице 4.

*Таблица 4 – Алгоритм деструктора класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | инициализация целочисленной переменной i значением 0 | 2 |
| 2 | i < размера вектора children | Освобождение памяти выделенной для объекта children[i] с помощью оператора функции delete | 3 |
| | | | ∅ |
| 3 | | i += 1 | 2 |

## 3.4 Алгоритм метода get_name класса cl_base

Функционал: Возвращает значение строчного поля name.

Параметры: .

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 5.

*Таблица 5 – Алгоритм метода get_name класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Возврат значения поля name | ∅ |

## 3.5 Алгоритм метода setName класса cl_base

Функционал: Производит валидацию на приеме имени объекта.

Параметры: name.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 6.

*Таблица 6 – Алгоритм метода setName класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | в векторе children, принадлежащем родительскому объекту, нет объектов с вводимым именем | Присвоение значения параметра name значению поля name текущего объекта | 2 |
| | | Возврат false | ∅ |
| 2 | | Возврат true | ∅ |

## 3.6 Алгоритм метода print_names класса cl_base

Функционал: Выводит наименования объектов в дереве иерархии слева направо и сверху вниз.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

*Таблица 7 – Алгоритм метода print_names класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | данный объект имеет дочерние объекты | Вывод перехода на новую строку и имени объекта | 2 |
| | | | ∅ |
| 2 | | Инициализация переменной i значением 0 | 3 |
| 3 | i < размера вектора children | Вывод двух пробелов и имени объекта children[i] | 4 |
| | | | 5 |
| 4 | | i += 1 | 3 |
| 5 | | Вызов метода print_names для последнего объекта в массиве children | ∅ |

## 3.7 Алгоритм метода get_parent класса cl_base

Функционал: Возвращает адрес поля parent.

Параметры: .

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 8.

*Таблица 8 – Алгоритм метода get_parent класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Возврат адреса поля parent текущего объекта | ∅ |

## 3.8 Алгоритм метода get_child_by_name класса cl_base

Функционал: Возвращает указатель на объект с заданным в параметрах именем.

Параметры: string name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 9.

*Таблица 9 – Алгоритм метода get_child_by_name класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Инициализация переменной i значением 0 | 2 |
| 2 | i < длины вектора children | | 3 |
| | | | 5 |
| 3 | Значения поля name текущего элемента контейнера vecOfChildrens равно значению параметра name | Возврат значения объекта children[i] | ∅ |
| | | | 4 |
| 4 | | i += 1 | 2 |
| 5 | | Возврат nullptr | ∅ |

## 3.9 Алгоритм конструктора класса application

Функционал: Конструктор класса cl_application.

Параметры: cl_base* ptr_parent.

Алгоритм конструктора представлен в таблице 10.

*Таблица 10 – Алгоритм конструктора класса application*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Вызов конструктора базового класса с аргументами , которые являются аргументами и для конструктора application | ∅ |

## 3.10 Алгоритм метода build_tree_objects класса application

Функционал: Строит дерево иерархии.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

*Таблица 11 – Алгоритм метода build_tree_objects класса application*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Инициализация указателя par на объект класса cl_base значением указателя на данный объект | 2 |
| 2 | | Инициализация указателя sub на объект класса cl_base значением nullptr | 3 |
| 3 | | Объявление двух переменных строкового типа rname и cname | 4 |
| 4 | | Ввод значения переменной rname | 5 |
| 5 | | Вызов метода setName, принимающего в качестве аргумента переменную rname | 6 |
| 6 | true = true | | 7 |
| | | | ∅ |
| 7 | | Ввод значений переменных rname и cname | 8 |
| 8 | rname = cname | | ∅ |
| | | | 9 |
| 9 | sub != nullptr и rname не равно результату вызова метода get_name переменной sub | par = sub | 10 |
| | | | 10 |
| 10 | у объекта par нет дочерних объектов с именем cname | Создание нового объекта sub класса cl_node(par, cname) с помощью оператора функции new | 6 |
| | | | 6 |

## 3.11 Алгоритм метода exec_app класса application

Функционал: Запуск системы.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 12.

*Таблица 12 – Алгоритм метода exec_app класса application*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Вывод результата работы метода get_name() | 2 |
| 2 | | Вызов метода print_names() | 3 |
| 3 | | Возврат 0 | ∅ |

## 3.12 Алгоритм конструктора класса cl_node

Функционал: Создание объекта элемента дерева.

Параметры: cl_base* ptr_parent.

Алгоритм конструктора представлен в таблице 13.

*Таблица 13 – Алгоритм конструктора класса cl_node*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | Вызов параметризированного конструктора класса cl_node и передача в<br>него в качестве параметра значение параметра ptr_parent | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-7.

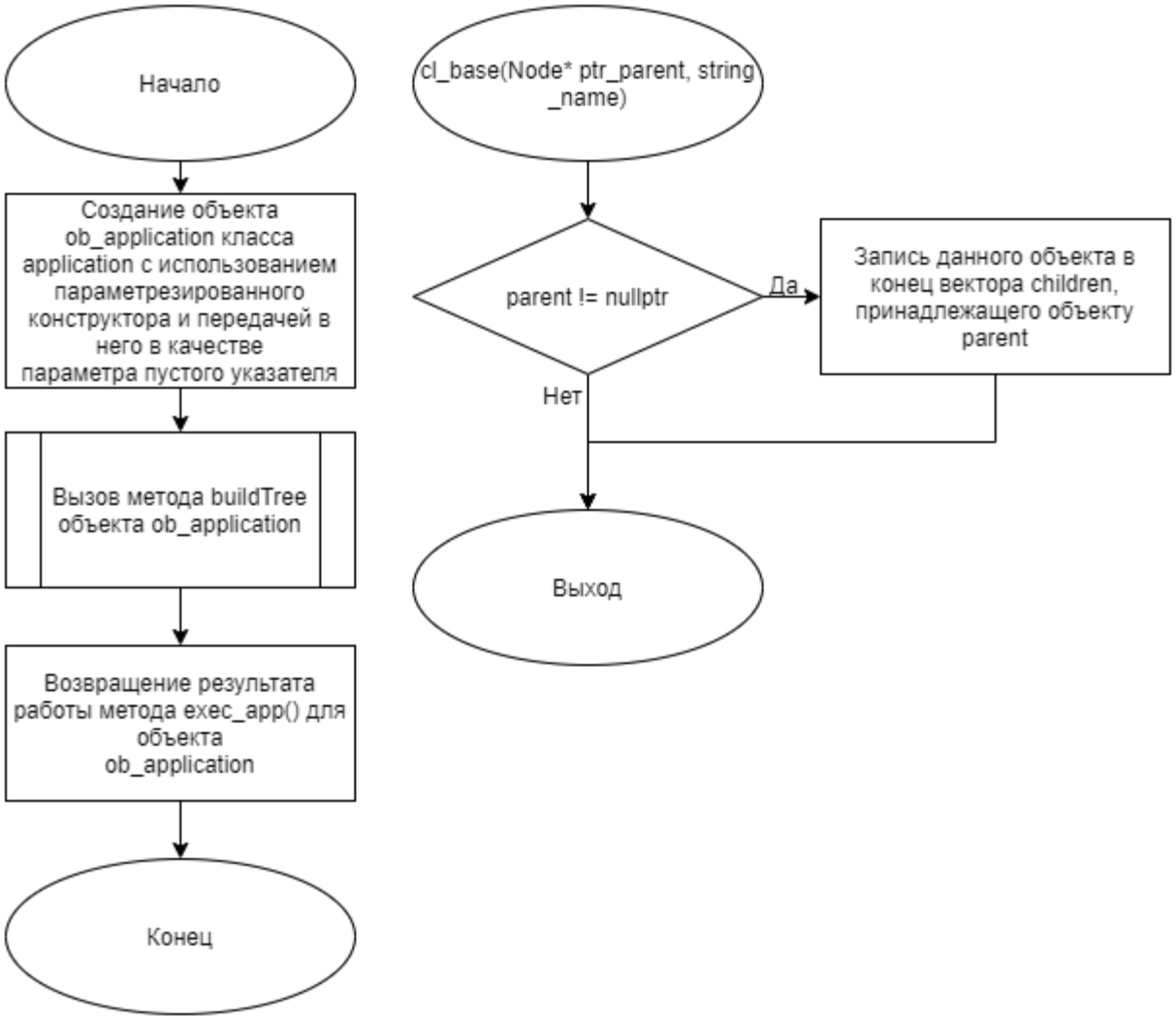


**Рисунок 1 – Блок-схема алгоритма**

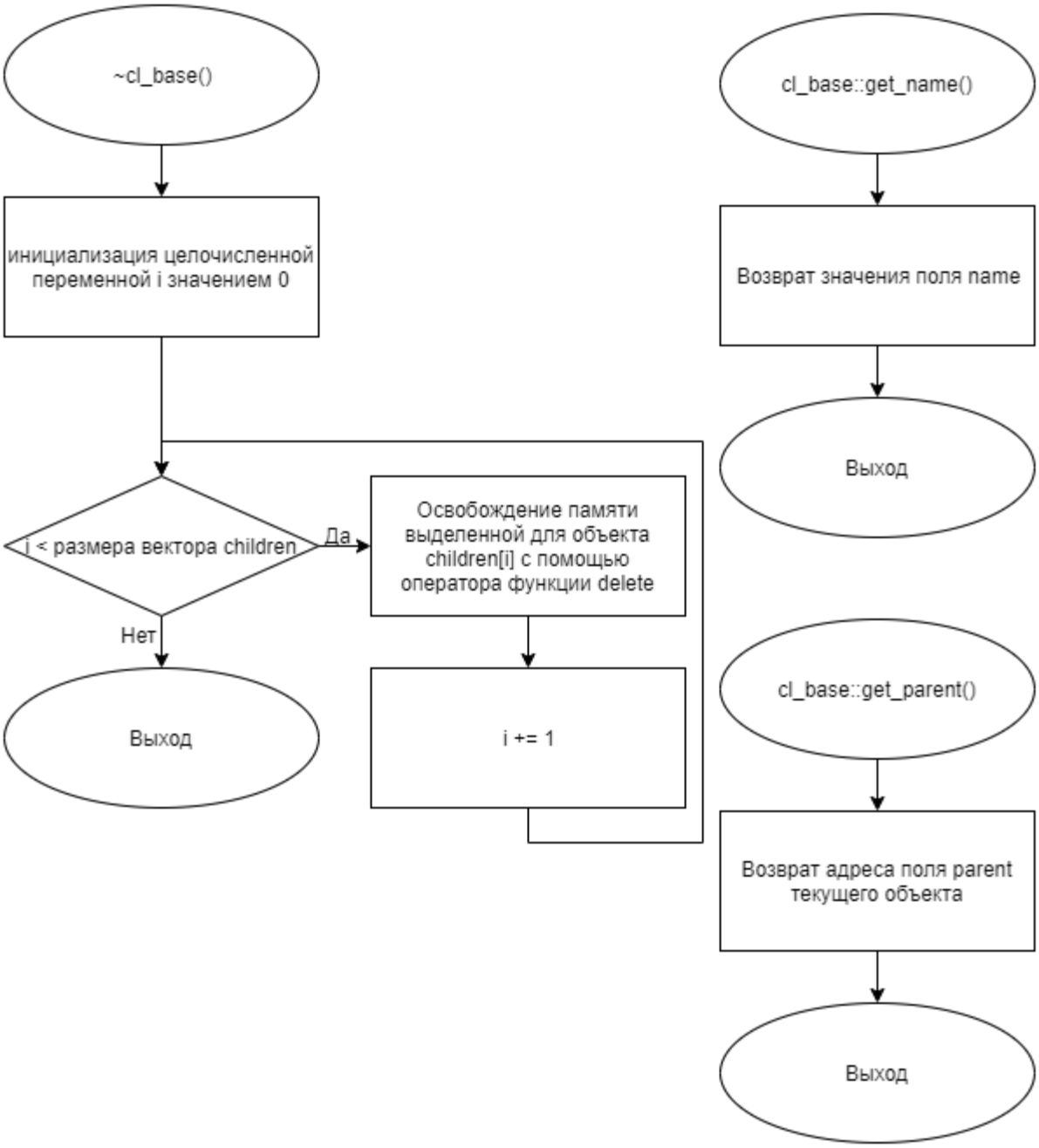


**Рисунок 2 – Блок-схема алгоритма**

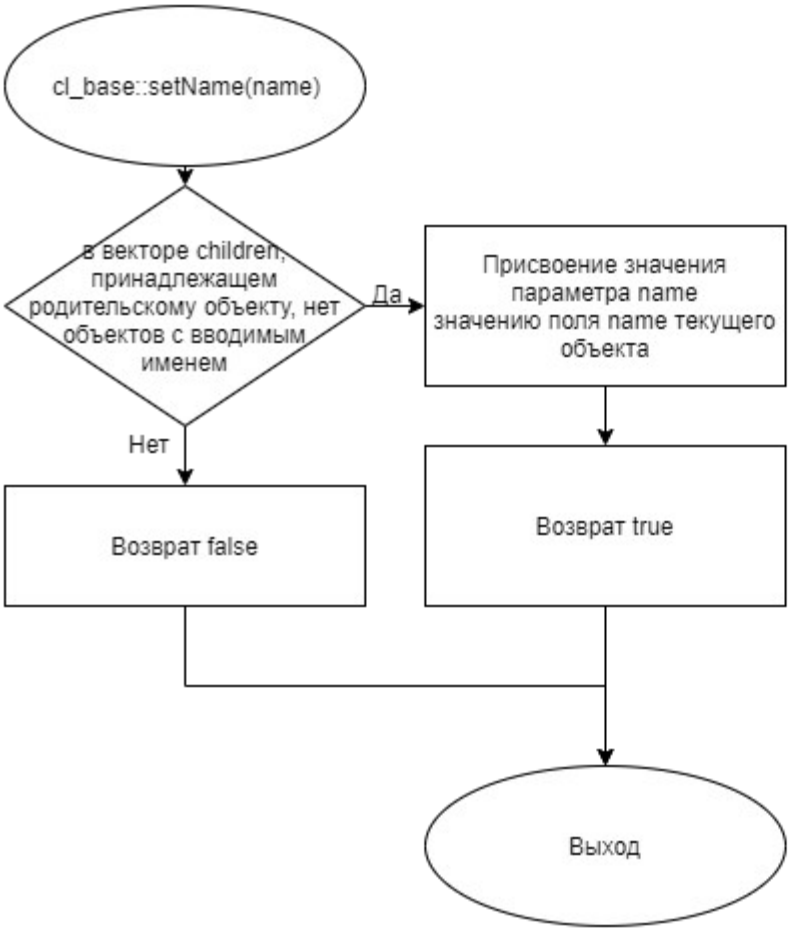


**Рисунок 3 – Блок-схема алгоритма**

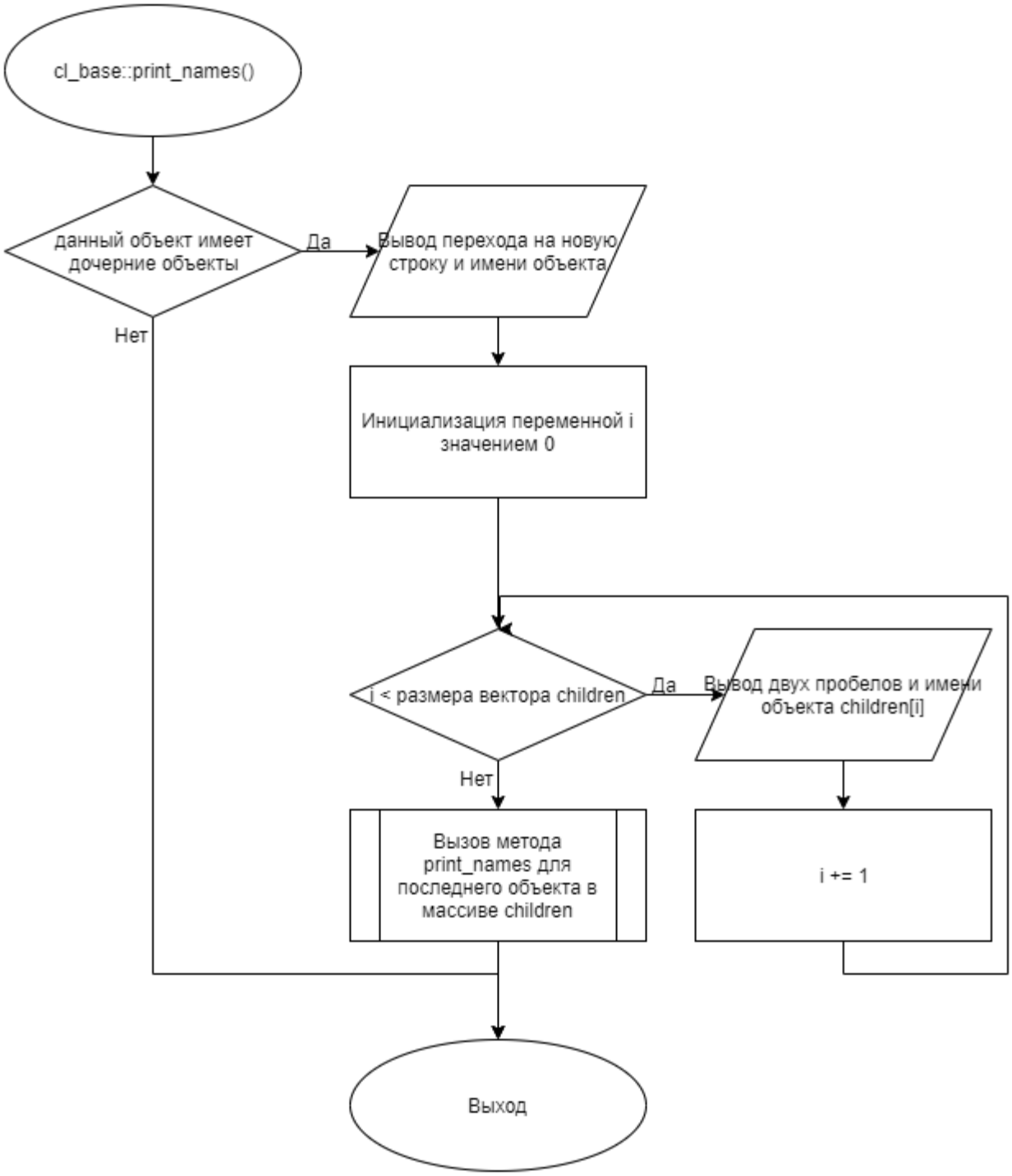


**Рисунок 4 – Блок-схема алгоритма**

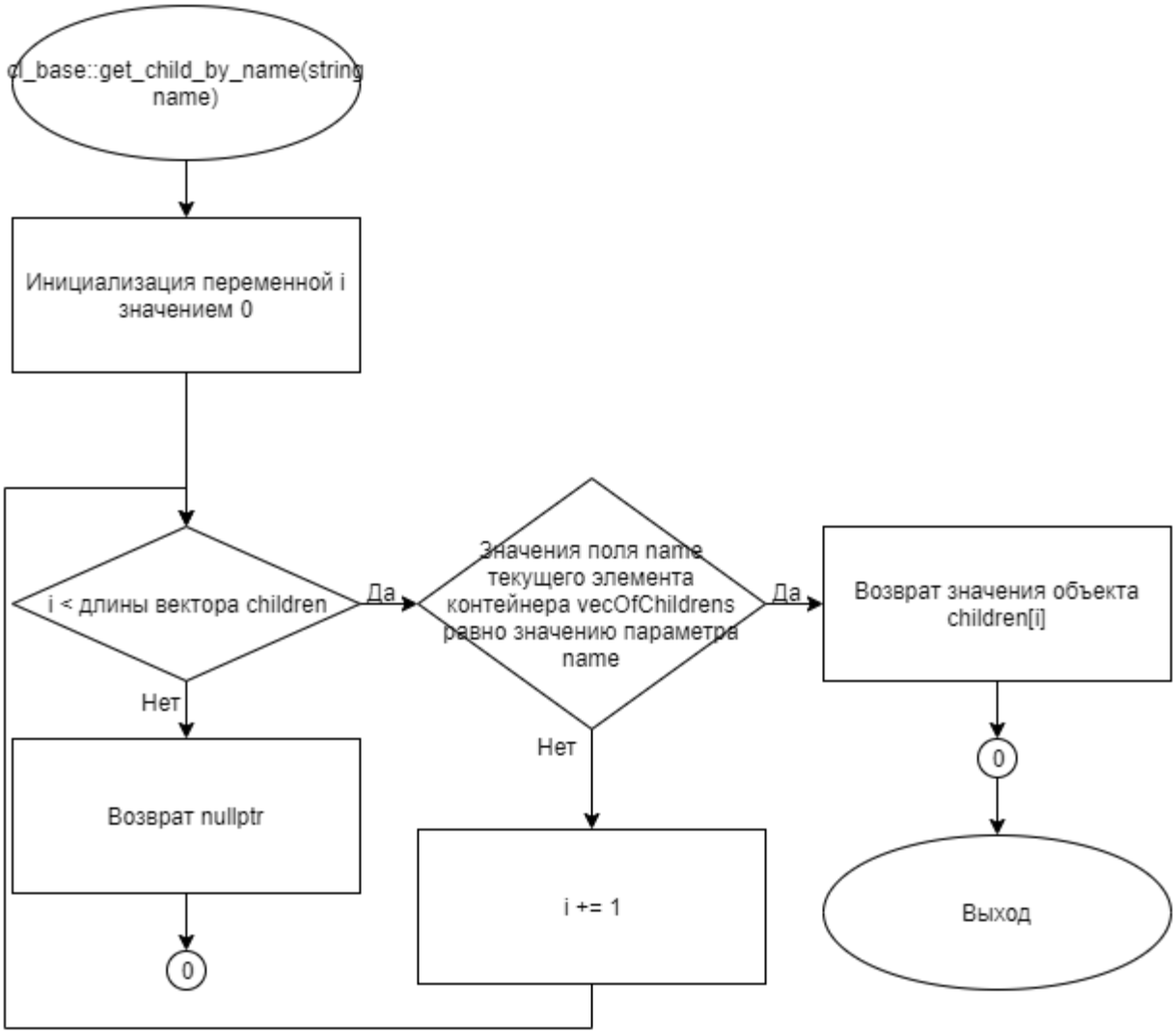


**Рисунок 5 – Блок-схема алгоритма**

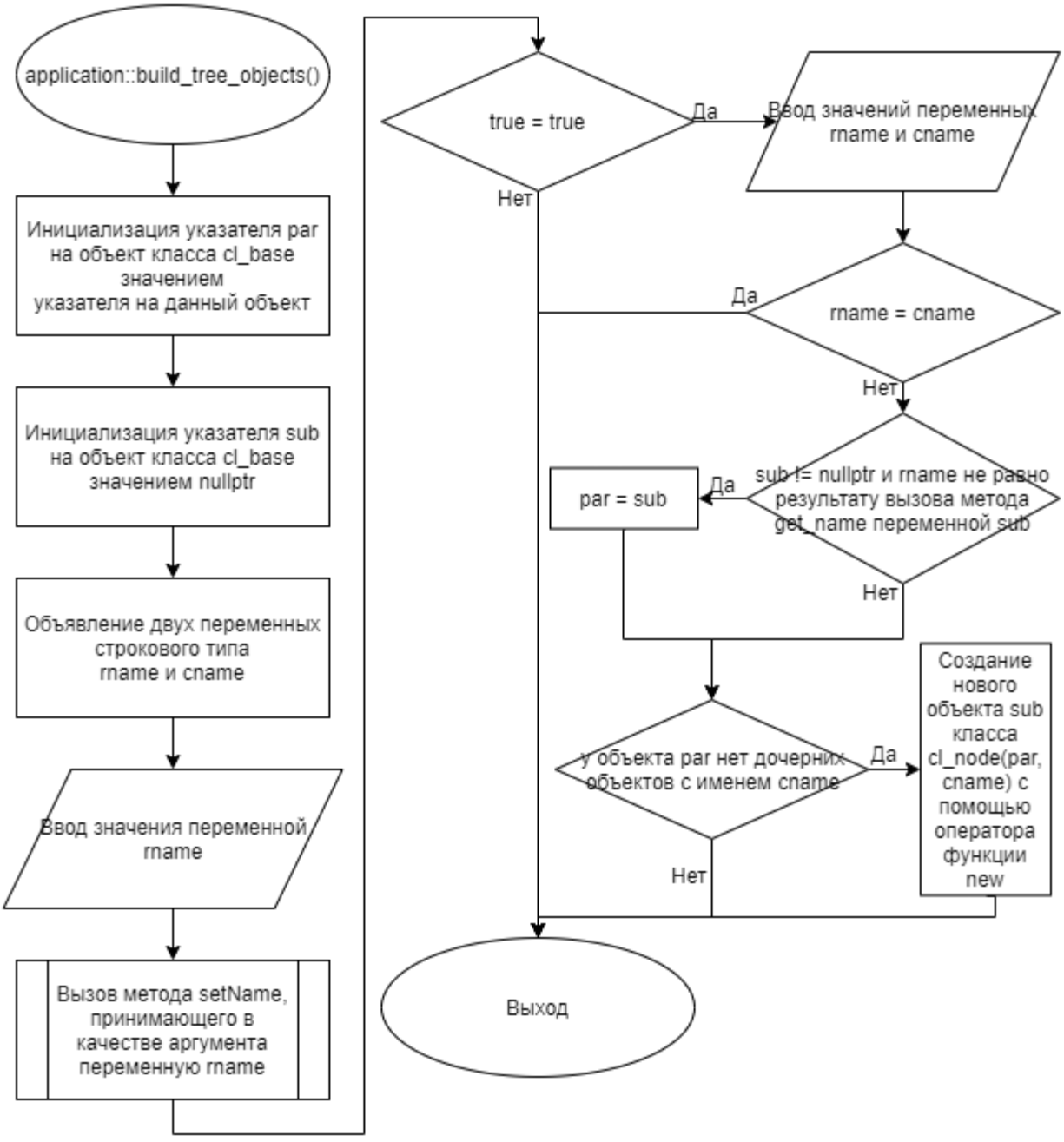


**Рисунок 6 – Блок-схема алгоритма**

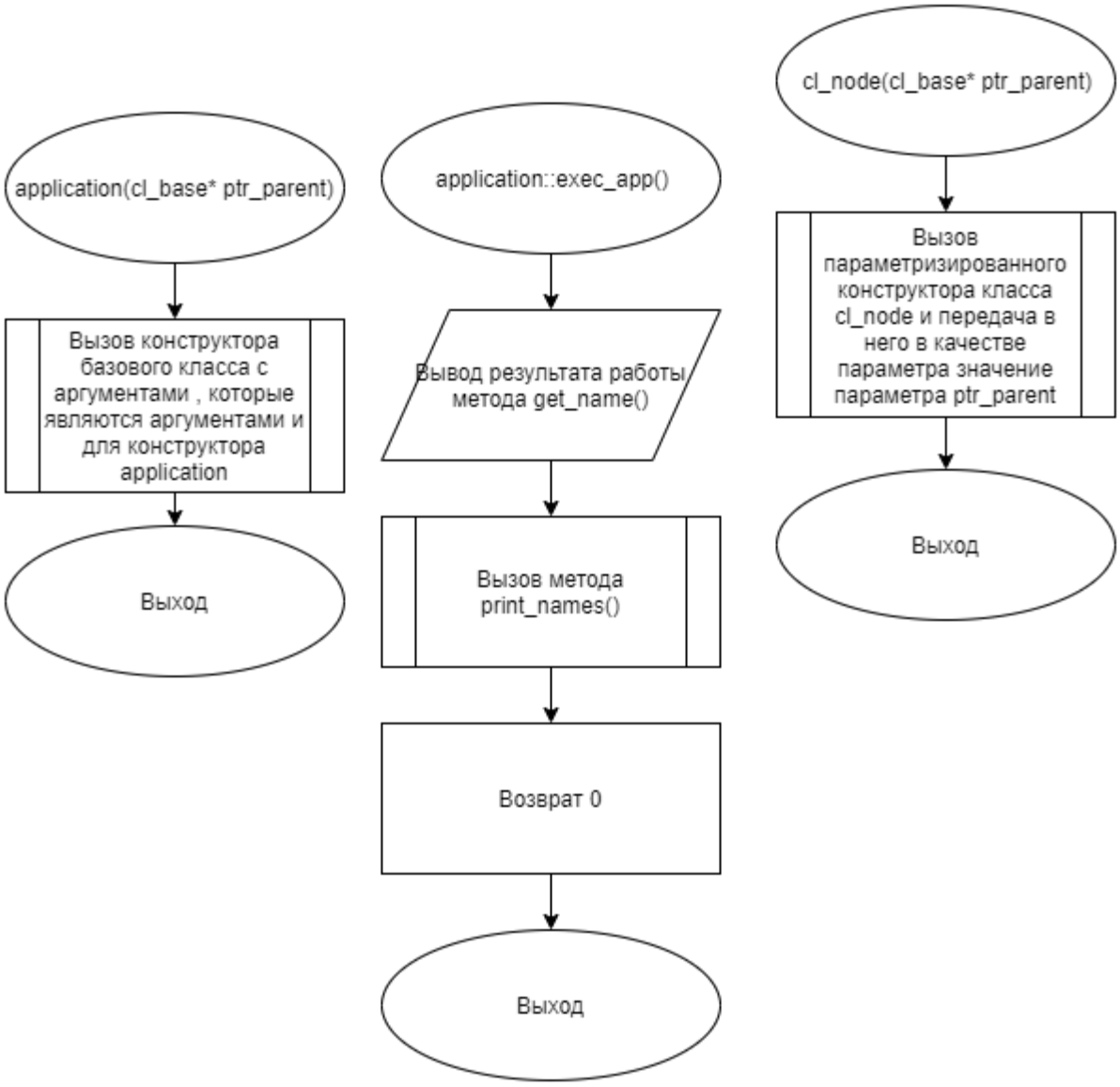


**Рисунок 7 – Блок-схема алгоритма**

# 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

## 5.1 Файл application.cpp

*Листинг 1 – application.cpp*

```
#include "application.h"

application::application(cl_base* parent): cl_base(parent) {}

int application::exec_app() {
      cout << get_name();
      print_names();
      return (0);
}

void application::build_tree_objects() {
      cl_base* par = this;
      cl_base* sub = nullptr;
      string rname, cname;
      cin >> rname;
      setName(rname);
      while (true) {
            cin >> rname >> cname;
            if (rname==cname) {
                  break;
            }
            if (sub != nullptr && rname == sub->get_name())
            {
                  par = sub;
            }
            if  (par->get_child_by_name(cname)  ==  nullptr  &&  rname  ==  par-
>get_name()) {
                  sub = new cl_node(par, cname);
            }
      }
}
```

## 5.2 Файл application.h

*Листинг 2 – application.h*

```
#ifndef __APPLICATION__H
```

```
#define __APPLICATION__H

#include "cl_base.h"
#include "cl_node.h"
class application: public cl_base {
public:
	application(cl_base* parent);
	void build_tree_objects();
	int exec_app();
};


#endif
```

## 5.3 Файл cl_base.cpp

*Листинг 3 – cl_base.cpp*

```
#include "cl_base.h"

cl_base::cl_base(cl_base* parent, string name): parent(parent), name(name) {
	if (parent != nullptr) {
		parent->children.push_back(this);
	}
}
cl_base::~cl_base() {
	for (int i = 0; i < children.size(); i++) delete children[i];
}

string cl_base::get_name() const {return name;}
cl_base* cl_base::get_parent() const {return parent;}

bool cl_base::setName(string name1) {
	if (get_parent() != nullptr && get_parent() -> get_child_by_name(name1) !=
nullptr) {
		return false;
	}
	name = name1;
	return true;
}

void cl_base::print_names() const {
	if (children.size() > 0) {
		cout << endl << name;
		for (int i = 0; i < children.size(); i++) {cout<<"   "<<children[i]-
>name;}
		children[children.size() - 1] -> print_names();
	}
}

cl_base* cl_base::get_child_by_name(string name) {
	for (auto child: children) {
		if (child->name == name) return child;
```

```
	}
	return nullptr;
}
```

## 5.4 Файл cl_base.h

*Листинг 4 – cl_base.h*

```
#ifndef __CL_BASE__H
#define __CL_BASE__H

#include <iostream>
#include <vector>
#include <string>
using namespace std;
class cl_base {
private:
	cl_base* parent;
	vector <cl_base*> children;
	string name;
public:
	cl_base(cl_base* parent, string name = "Object_root");
	~cl_base();
	bool setName(string name);
	string get_name() const;
	cl_base* get_parent() const;
	void print_names() const ;
	cl_base* get_child_by_name(string name);
};


#endif
```

## 5.5 Файл cl_node.cpp

*Листинг 5 – cl_node.cpp*

```
#include "cl_node.h"
cl_node::cl_node(cl_base* parent, 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"
class cl_node: public cl_base {
public:
cl_node(cl_base* parent, string name);
};


#endif
```

## 5.7 Файл main.cpp

*Листинг 7 – main.cpp*

```
#include "application.h"
int main() {
      application ob_application(nullptr);
      ob_application.build_tree_objects();
      return (ob_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<br>Object_root Object_1<br>Object_root Object_2<br>Object_root Object_3<br>Object_3 Object_4<br>Object_3 Object_5<br>Object_6 Object_6 | Object_root<br>Object_root Object_1<br>Object_2 Object_3<br>Object_3 Object_4<br>Object_5 | Object_root<br>Object_root Object_1<br>Object_2 Object_3<br>Object_3 Object_4<br>Object_5 |
| Object_root<br>Object_root object_1<br>Object_root object_2<br>object_1 object_3<br>object_1 object_4<br>object_1 object_4<br>object_1 object_5<br>object_1 object_2<br>object_2 object_13<br>object_2 object_14<br>object_2 object_14<br>object_2 object_15<br>object_3 object_3 | Object_root<br>Object_root object_1<br>object_2<br>object_2 object_13<br>object_14 object_15 | Object_root<br>Object_root object_1<br>object_2<br>object_2 object_13<br>object_14 object_15 |

# 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).