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 понадобится:

- cout - объект потока вывода
- cin - объект потока ввода
- if - условный оператор
- while, for - цикл
- библиотека string, vector
- указатель this

**Класс cl_base базовый класс**

Поля:

скрытые элементы:

string object_name - наименование объекта

cl_base* p_parent - указатель на головной объект

открытые элементы:

vector <cl_base*> children - указатели на подчиненные объекты

vector <cl_base*> :: iterator it_child

методы:

открытые:

cl_base(cl_base* parent=nullptr, string name="") - конструктор класса

bool set_object_name(string object_name) - редактирование имени объекта

string get_object_name() - получение имени объекта

cl_base* get_parent() - получение указателя на головной объект текущего объекта

void show_tree() - вывод наименований объектов в дереве иерархии

cl_base* get_child(string child_name) - получение указателя на

непосредственно подчиненный объект по его имени

**Класс cl_application наследует класс cl_base**

Поля:

методы:

cl_application() - конструктор класса

void build_tree_objects() - построение иерархии классов

int exec_app() - вывод в консоль иерархии классов

**Класс cl_1 наследует класс cl_base**

Поля:

методы:

**Класс cl_2 наследует класс cl_base**

Поля:

методы:

# 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: главный метод программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 1.

*Таблица 1 – Алгоритм функции main*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | создание объекта класса | 2 |
| 2 | | вызов метода build_tree_objects | 3 |
| 3 | | вызов метода exec_app | ∅ |

## 3.2 Алгоритм конструктора класса cl_application

Функционал: создание корневого объекта.

Параметры: нет.

Алгоритм конструктора представлен в таблице 2.

*Таблица 2 – Алгоритм конструктора класса cl_application*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | создание корневого объекта | ∅ |

## 3.3 Алгоритм метода build_tree_objects класса cl_application

Функционал: создание иерархии объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

*Таблица 3 – Алгоритм метода build_tree_objects класса cl_application*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | инициализация строковых переменных root_name, child_name | 2 |
| 2 | | ввод root_name | 3 |
| 3 | | создание массива created, который хранит ссылки | 4 |
| 4 | | изменения имени корневого объекта на root_name | 5 |
| 5 | | добавление корневого объекта в массив created | 6 |
| 6 | | ввод root_name, child_name | 7 |
| 7 | root_name не равен child_name | | 8 |
| | | | ∅ |
| 8 | c принадлежит created | | 9 |
| | | | 7 |
| 9 | c->get_object_name() = root_name и у с нет среди подчинённых объектов объекта с именем = child_name | создание нового объекта класса с параметрами ввиде: c, child_name | 10 |
| | | | 8 |
| 10 | количество элементов в created чётно | создание нового объекта класса cl_2 с параметрами ввиде: c, child_name | 11 |
| | | создание нового объекта класса cl_1 с параметрами | 11 |

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| | | ввиде: c, child_name | |
| 11 | | добавление нового объекта в массив created | 7 |

## 3.4 Алгоритм метода exec_app класса cl_application

Функционал: вывод иерархии объектов.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 4.

*Таблица 4 – Алгоритм метода exec_app класса cl_application*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | вызов метода show_tree для корневого объекта | ∅ |

## 3.5 Алгоритм конструктора класса cl_base

Функционал: создание связи с родительским объектом и установка имени текущего.

Параметры: cl_base* parent, string name.

Алгоритм конструктора представлен в таблице 5.

*Таблица 5 – Алгоритм конструктора класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | скрытое свойство p_parent равно parent | 2 |
| 2 | | скрытое свойство object_name равно name | 3 |
| 3 | parent != nullptr | добавить текущий объект к списку подчинённых объектов parent | ∅ |

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| | | | ∅ |

## 3.6 Алгоритм метода get_object_name класса cl_base

Функционал: возвращение имя текущего объекта.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 6.

*Таблица 6 – Алгоритм метода get_object_name класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | возвращение имя текущего объекта | ∅ |

## 3.7 Алгоритм метода get_parent класса cl_base

Функционал: возвращение указателя на головной объект текущего объекта.

Параметры: нет.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 7.

*Таблица 7 – Алгоритм метода get_parent класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | возвращение указателя на головной объект текущего объекта | ∅ |

## 3.8 Алгоритм метода set_object_name класса cl_base

Функционал: изменение имени текущего объекта.

Параметры: string s.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 8.

*Таблица 8 – Алгоритм метода set_object_name класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | у текущего объекта нет указателя на головной объект | имя текущего объекта = s | 2 |
| | | | 3 |
| 2 | | возвращение true | ∅ |
| 3 | | it_child = children.begin() | 4 |
| 4 | it_child != children.end() | | 5 |
| | | | 6 |
| 5 | (*it_child)->get_object_name и (*it_child) != текущему объекту | возвращение false | ∅ |
| | | it_child++ | 4 |
| 6 | | имя текущего объекта = s | 7 |
| 7 | | возвращение true | ∅ |

## 3.9 Алгоритм метода get_child класса cl_base

Функционал: поиск подчинённого объекта по имени.

Параметры: string child_name.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 9.

*Таблица 9 – Алгоритм метода get_child класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | | it_child = children.begin() | 2 |
| 2 | it_child != children.end() | | 3 |

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| | | | 4 |
| 3 | (*it_child)->get_object_name() = child_name | возвращение (*it_child) | ∅ |
| | | it_child++ | 2 |
| 4 | | возвращение nullptr | ∅ |

## 3.10 Алгоритм метода show_tree класса cl_base

Функционал: вывод иерархии объекта.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 10.

*Таблица 10 – Алгоритм метода show_tree класса cl_base*

| № | Предикат | Действия | № перехода |
|---|---|---|---|
| 1 | children.size() > 0 | | 2 |
| | | | ∅ |
| 2 | текущий объект не имеет головного объекта | вывод имя текущего объекта | 3 |
| | | | 3 |
| 3 | | вывод имя текущего объекта | 4 |
| 4 | | it_child = children.begin() | 5 |
| 5 | it_child != children.end() | вывод (*it_child)->get_object_name | 5 |
| | | | 6 |
| 6 | | it_child = children.begin() | 7 |
| 7 | it_child != children.end() | вызов метода (*it_child)->show_tree() | 7 |
| | | | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-6.

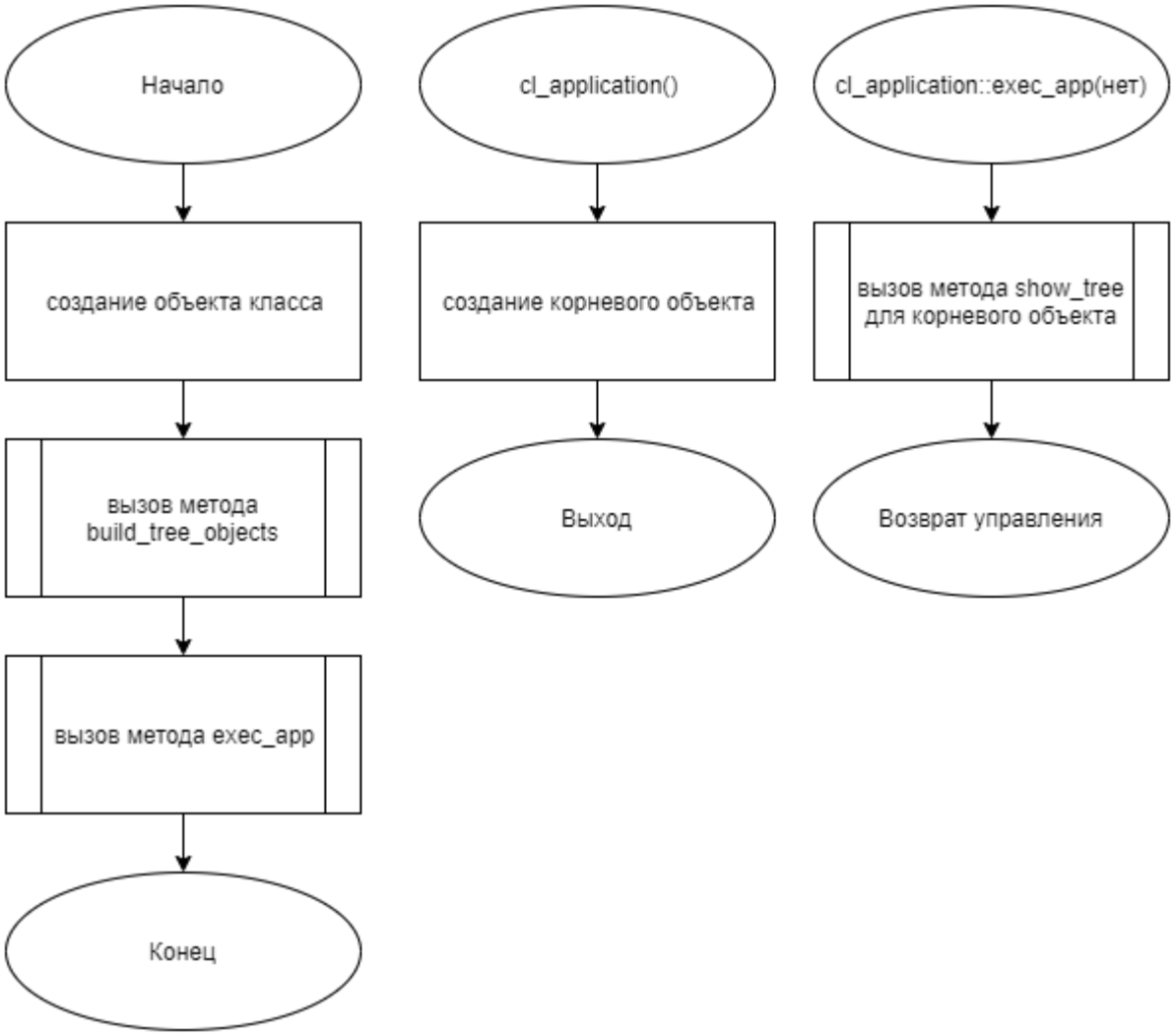


**Рисунок 1 – Блок-схема алгоритма**

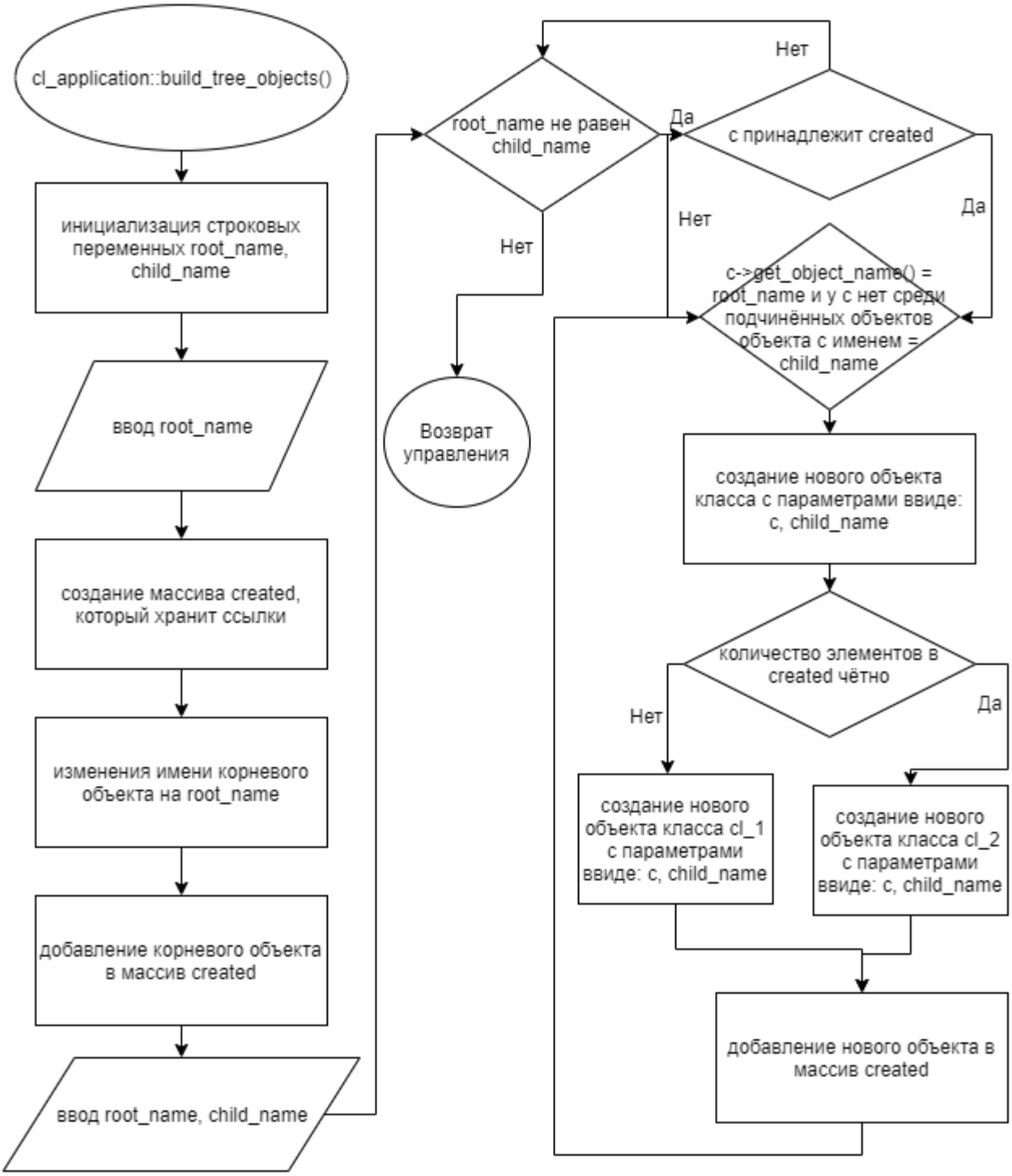


**Рисунок 2 – Блок-схема алгоритма**

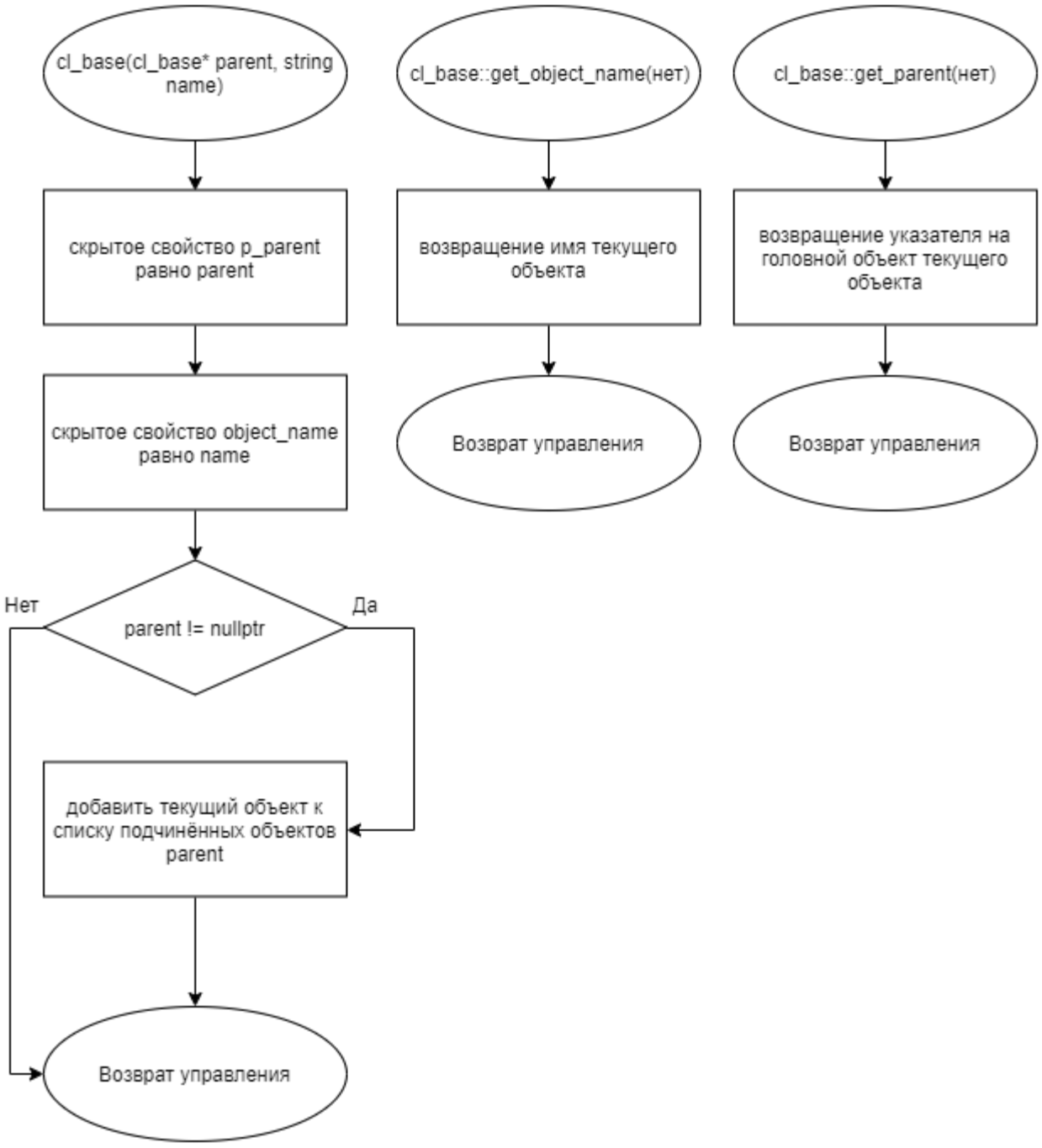


**Рисунок 3 – Блок-схема алгоритма**

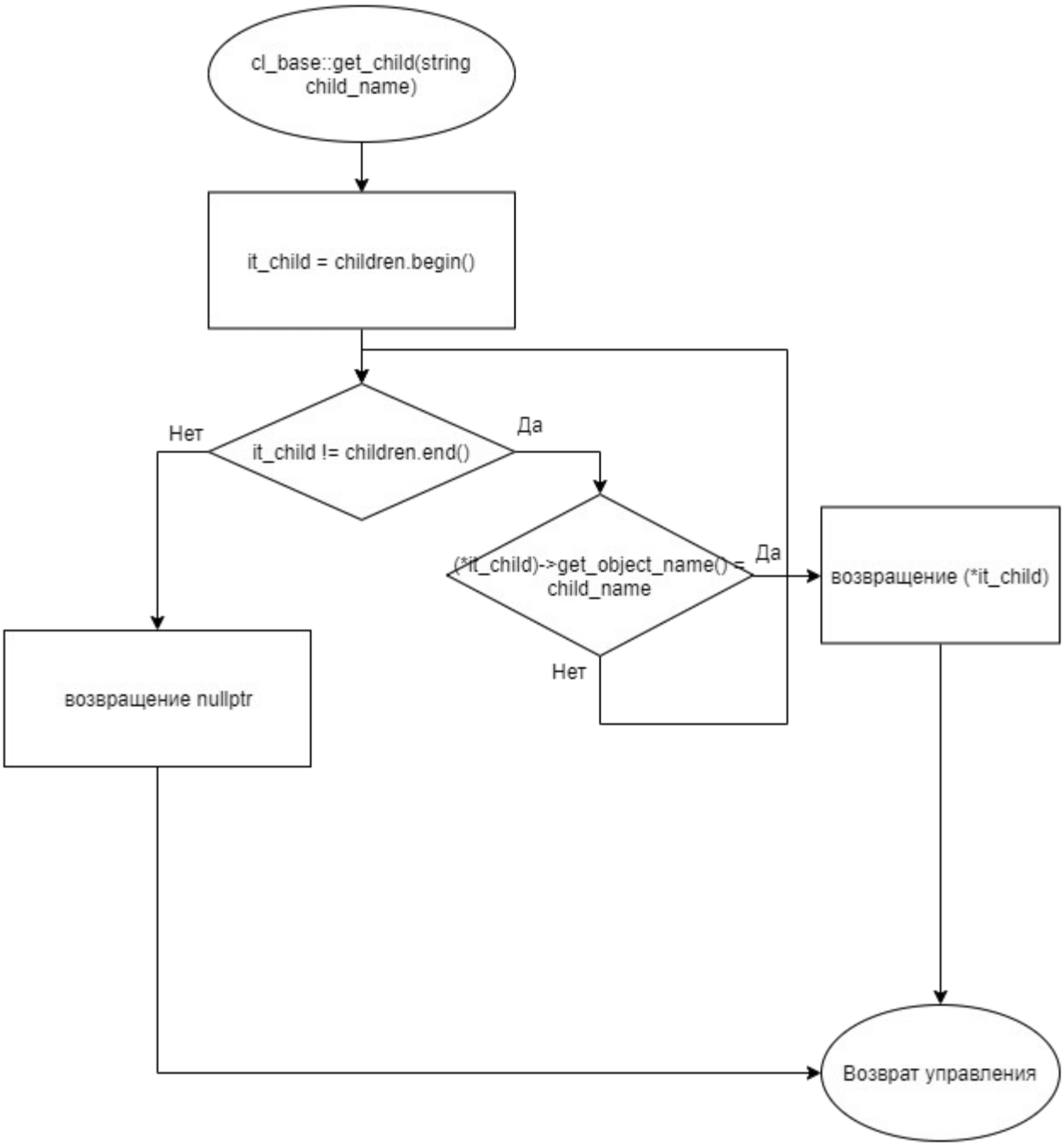


**Рисунок 4 – Блок-схема алгоритма**

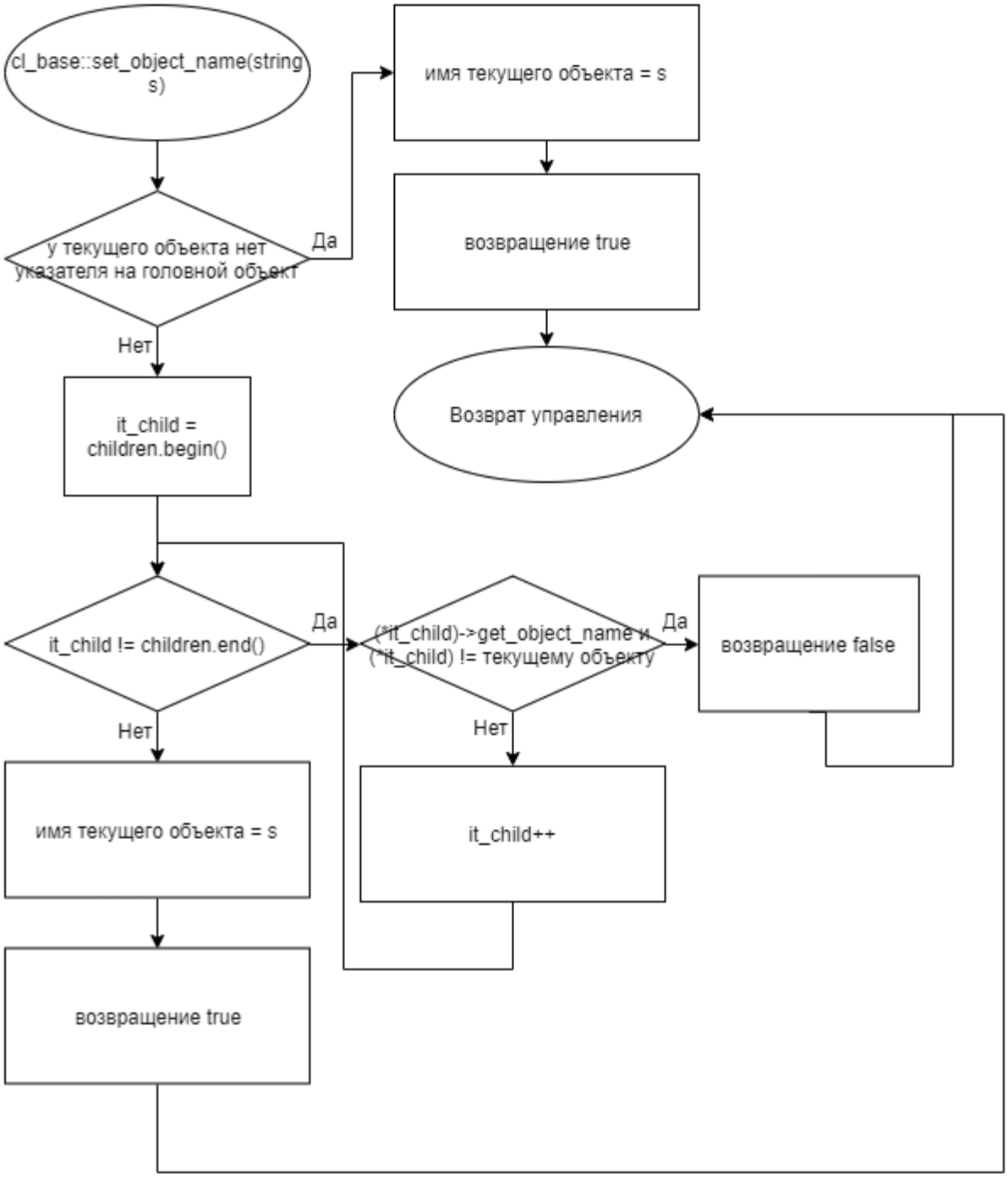


**Рисунок 5 – Блок-схема алгоритма**

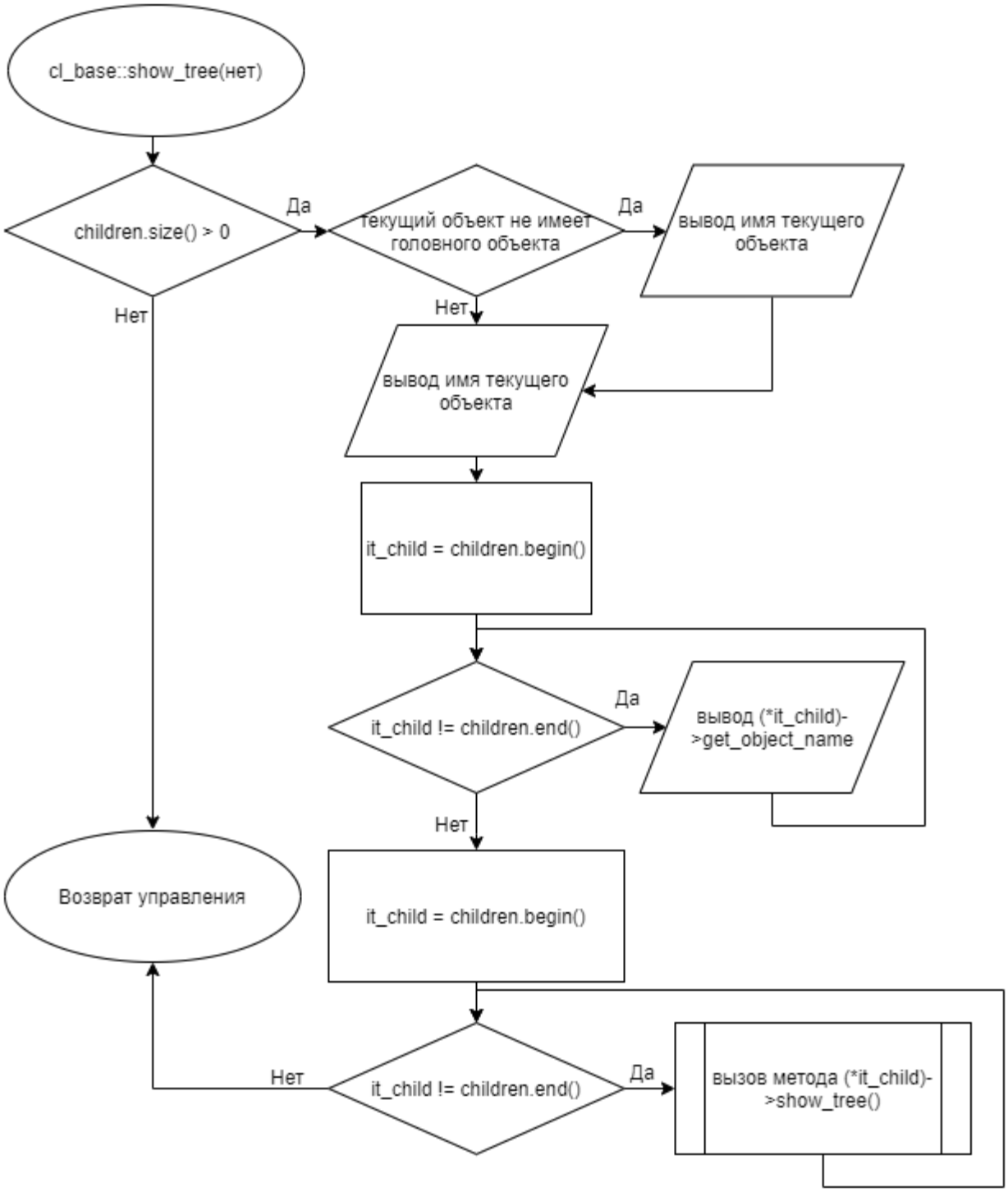


**Рисунок 6 – Блок-схема алгоритма**

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

using namespace std;

cl_1::cl_1(cl_base* parent, string name) : cl_base(parent, name) {};
```

## 5.2 Файл cl_1.h

*Листинг 2 – cl_1.h*

```
#ifndef __CL_1__H
#define __CL_1__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_1 : public cl_base
{
public:
	cl_1(cl_base* parent=nullptr, string name="");
};

#endif
```

## 5.3 Файл cl_2.cpp

*Листинг 3 – cl_2.cpp*

```
#include "cl_2.h"
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

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

## 5.4 Файл cl_2.h

*Листинг 4 – cl_2.h*

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_2 : public cl_base
{
public:
	cl_2(cl_base* parent=nullptr, string name="");
};

#endif
```

## 5.5 Файл cl_application.cpp

*Листинг 5 – cl_application.cpp*

```
#include "cl_application.h"
#include "cl_1.h"
#include "cl_2.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
cl_application::cl_application() : cl_base(nullptr, "") {};

void cl_application::build_tree_objects()
{
	string root_name, child_name;
	cin >> root_name;
	vector <cl_base*> created;
	this->set_object_name(root_name);
	created.push_back(this);
	cin >> root_name >> child_name;
	while (root_name != child_name)
	{
		for (auto c: created)
		{
			if	(c->get_object_name()	==	root_name	&&	!c-
>get_child(child_name))
			{
				if (c->get_parent() && c->get_parent()->children.size() !=
0 && *(c->get_parent()->children.end() - 1) != c)
				{
					break;
				}
				if (created.size() % 2 == 0)
				{
					cl_base* n = new cl_2(c, child_name);
					created.insert(created.begin(), n);
					break;
				}
				else
				{
					cl_base* n = new cl_1(c, child_name);
					created.insert(created.begin(), n);
					break;
				}
			}
		}
		cin >> root_name >> child_name;
	}
}

int cl_application::exec_app()
{
	this->show_tree();
	return 0;
}
```

## 5.6 Файл cl_application.h

*Листинг 6 – cl_application.h*

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
```

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

using namespace std;

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

#endif
```

## 5.7 Файл cl_base.cpp

*Листинг 7 – cl_base.cpp*

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

using namespace std;

cl_base::cl_base(cl_base*	parent,	string	name):	p_parent(parent),
object_name(name)
{
	if (parent != nullptr)
	{
		parent->children.push_back(this);
	}
}

bool cl_base::set_object_name(string s)
{
	if (!p_parent)
	{
		object_name = s;
		return true;
	}

	it_child = children.begin();

	while (it_child != children.end())
	{
		if (((*it_child)->get_object_name() == s) && ((*it_child) != this))
		{
			return false;
		}
```

```
            it_child++;
      }

      object_name = s;
      return true;
}

string cl_base::get_object_name()
{
      return object_name;
}

cl_base* cl_base::get_parent()
{
      return p_parent;
}

void cl_base::show_tree()
{
      if (children.size() > 0)
      {
            if (!p_parent)
            {
                  cout << object_name;
            }

            cout << endl;
            cout << object_name;

            it_child = children.begin();

            while (it_child != children.end())
            {
                  cout << "  " << (*it_child)->get_object_name();
                  it_child++;
            }

            it_child = children.begin();

            while (it_child != children.end())
            {
                  (*it_child)->show_tree();
                  it_child++;
            }
      }
}

cl_base* cl_base::get_child(string child_name)
{
      it_child = children.begin();

      while (it_child != children.end())
      {
            if ((*it_child)->get_object_name() == child_name)
            {
                  return (*it_child);
            }
```

```
            it_child++;
      }

      return nullptr;
}
```

## 5.8 Файл cl_base.h

*Листинг 8 – cl_base.h*

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

using namespace std;

class cl_base
{
private:
      string object_name;
      cl_base* p_parent;
      //vector <cl_base*> children;
      //vector <cl_base*> :: iterator it_child;
public:
      vector <cl_base*> children;
      vector <cl_base*> :: iterator it_child;
      cl_base(cl_base* parent=nullptr, string name="");
      bool set_object_name(string object_name);
      string get_object_name();
      cl_base* get_parent();
      void show_tree();
      cl_base* get_child(string child_name);
};

#endif
```

## 5.9 Файл main.cpp

*Листинг 9 – main.cpp*

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "cl_application.h"

int main()
{
```

```
        cl_application ob_cl_application;
        ob_cl_application.build_tree_objects();
        ob_cl_application.exec_app();
        return(0);
}
```

# 6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 11.

*Таблица 11 – 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 |

# 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).