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.

Создать базовый класс со следующими элементами:

Свойства:

- наименование объекта (строкового типа);
- указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно 0);
- массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.

Функционал:

- параметризированный конструктор с параметрами: указатель на головной объект в дереве иерархии и наименование объекта (имеет значение по умолчанию);
- метод определения имени объекта;
- метод получения имени объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод переопределения головного объекта для текущего в дереве иерархии (не допускается задание головного объекта для корневого и появление второго корневого объекта);

- метод получения указателя на головной объект текущего 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 программы-системы, изделия);

− 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. Каждый объект имеет уникальное имя.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
int main()
{
   cl_application  ob_cl_application ( nullptr );
   ob_cl_application.bild_tree_objects ( );  // 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.

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.

Примерввода

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 выполнения нам потребуются:

объектов потока ввода и вывода (cin, cout)

стандартные функции библиотеки <vector>

объекты класса string библиотеки <string>

объекты класса cl_base

объекты класса cl_devider

объект obj класса cl_application,

Класс cl_base:

1. Свойства/поля:

- Поле отвечающее за наименование объекта

1. Наименование - name;
2. Тип данных - string;
1. Модификатор доступа - private;

- Поле отвечающее за указатель на головной объект (используется для хранения головного объекта при построении дерева объектов):

Наименование - parent;

Тип данных -cl_base*;

Модификатор доступа - private;

- Поле отвечающее за хранение указателей на иерархически подчиненных объектов (массив указателей объектов-потомков):

Наименование - child

Тип данных - vector <cl_base*>;

Модификатор доступа - private;

2. Методы:

Конструктор :

Функционал- параметризированный конструктор с параметрами: указатель на головной объект в дереве иерархии и наименованием объекта (имеет значение по умолчанию). Присваивает полям класса соответствующие значения. Добавляет ненулевой указаетль в массив указателей потомков головного объекта;

Деструктор:

Функционал - удаляет массив потомков головного объекта поэлементно;

Метод input_name

Функционал - метод с параметром типа string, значение которого присваивается полю name, для определения имени объекта;

Метод output_name

Функционал - позволяет получить имя объекта;

Метод output_tree

Функционал - выводит наименования объектов в дереве иерархии слева направо и сверху вниз;

Метод set_parent

Функционал - переопределяет головной объект для текущего в дереве иерархии, параметр указывает на новый головной объект;

Метод output_parent

Функционал - позволяет получить указатель на головной объект для текущего ;

Класс cl_application:

Свойства/поля:

Наследуется cl_base с модификатором доступа public;

Методы:

Конструктор

Функционал - в него передается в качестве параметра указатель типа Base*, который передается в качестве параметра конструктору базового класса;

Метод build_tree

Функционал - строит исходное дерево иерархии объектов;

Метод exec_app

Функционал - запускает приложение;

Класс cl_devider:

Свойства/поля:

Наследуется от класса cl_base с модификатором доступа public;

Методы:

Конструктор

Функционал - в него передается в качестве параметра указатель типа cl_base* и параметр типа string, которые передается в качестве параметров конструктору базового класса;

Таблица 1 – Иерархия наследования классов

Номер	Имя класса	Классы Наследники	Модификатор доступа	Номер класса	Описание	Комментарий
1	cl_base	cl_applications cl_devider	public public	2 3	Базовый Класс иерархии классов содержащий основные поля и	

					методы	
2	cl_applications				Класс корневого объекта	
3	cl_devider				Класс объектов подчинённых корневому объекту	

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: Создаёт объект класса cl_application.

Параметры: .

Возвращаемое значение: целочисленное значение.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта obj_application класса Application путем вызова конструктора с параметором nullptr	2
2		Вызов метода build_tree объекта obj_application	3
3		aВозвращение результата вызова метода exec_app объекта obj_application	∅

3.2 Алгоритм конструктора класса cl_base

Функционал: Присваивает полям объекта значение соответствующих параметров.

Параметры: cl_base* head_ptr - указатель на головной объект; string name - имя объекта.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Присвоение полю объекта name значения параметра name	2
2		Присвоение полю объекта parent значение параметра parent	3
3	Параметр parent ненулевой	Вызывает метод push_back по child головного объекта подавая ссылку на текущий головной обэъект в качестве параметра	∅
			2

3.3 Алгоритм деструктора класса cl_base

Функционал: Очищает массиво бъектов-потомков.

Параметры: нет.

Алгоритм деструктора представлен в таблице 4.

Таблица 4 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация целочисленной переменной i, имеющая значение 0	2
2	Значение параметра i < размера массива-потомков объектов	Удаление элемента массива child с индексом i	3
			∅
3		Прибавление единицы к значению переменной i	2

3.4 Алгоритм метода input_name класса cl_base

Функционал: Присваевает новое имя объекту.

Параметры: string name - Имя объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода input_name класса cl_base

№	Предикат	Действия	№ перехода
1		Присвоение полю name объекта значения параметра name	∅

3.5 Алгоритм метода output_name класса cl_base

Функционал: Получает имя объекта.

Параметры: нет.

Возвращаемое значение: string name - имя объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода output_name класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращает значения поля name имени объекта	∅

3.6 Алгоритм метода set_parent класса cl_base

Функционал: Переопределяет головной объект для текущего в деревеиерархии.

Параметры: cl_base* parent - указатель на головной объект.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_parent класса cl_base

№	Предикат	Действия	№ перехода
1	Значения текущего головного объекта this->parent и головного объекта не равны 0	инициализация целочисленной переменной i со значением 0	2
			∅
2	Значение переменной i < количества подчинённых текущекого головного объекта this->parent		3
			5
3	Значение элемента с индексом i массива объектов-потомков головного объекта равно указатель на текущий объект является элемент в списке подчинённых объектов с индексом i	Удаление элемента с индексом i из списка подчинённых объектов текущего головного объекта this->parent с помощью метода erase	4
			4
4		Прибавление единицы к значению переменной i	2
5		Присваивает свойству parent текущего объекта новое значение параметра parent	6
6		Добавление указателя текущего объекта в массив указателей потомков нового головного объекта this->parent применяя к свойству child нового головного объекта this-> parent метод push_back с указателем на текущий объект в качестве	∅

№	Предикат	Действия	№ перехода
		параметра	

3.7 Алгоритм метода output_tree класса cl_base

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

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода output_tree класса cl_base

№	Предикат	Действия	№ перехода
1	Количество подчиненных текущего головного объекта this->parent меньше 0	Выводит на экран значение свойства name текущего объекта	2
			5
2		инициализация целочисленной переменной i со значением 0	3
3	Значение переменной i < значение колисчества потомков текущего головного объекта	Вывод в консоль двух знаков пробела и значение name подчинённого объекта, указателем которой является элемент массива child с индексом i	4
			5
4		Прибавление единицы к значению переменной i	3

№	Предикат	Действия	№ перехода
5		Повторная инициализация целочисленной переменной i со значеним 0	6
6	У подчинённого child[i] существуют свои подчинённые	Вывод endl	7
			7
7	Значение i < размера массива объектовпотомков	Вызов метода output_tree по подчинённому объекту, указатель на который является элемент массива child по индексу i	8
			∅
8		Прибавление единицы к значению переменной i	7

3.8 Алгоритм метода output_parent класса cl_base

Функционал: Позволяет получить указатель головного объекта для текущего.

Параметры: нет.

Возвращаемое значение: cl_base* - Указатель на головной объект.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода output_parent класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращение значения поля parent	∅

3.9 Алгоритм конструктора класса cl__application

Функционал: Инициализирует поля объекта.

Параметры: cl_base* parent- указатель на головной объект, string name .

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl__application

№	Предикат	Действия	№ перехода
1		Вызов конструктора класса cl_base, которому передается в качестве параметров параметр parent	∅

3.10 Алгоритм метода exec_app класса cl_applicatilon

Функционал: Запуск Приложения.

Параметры: нет.

Возвращаемое значение: int, 0, код ошибки.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec_app класса cl_applicatilon

№	Предикат	Действия	№ перехода
1		Вывод в консоль результата вызова метода outPut_name текущего объекта	2
2		Вызов метода build_tree текущего объекта	3
3		Возвращение 0	∅

3.11 Алгоритм метода build_tree класса cl_application

Функционал: Строит дерево иерархии объектов класса Devider и объектакласса Application.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода build_tree класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление string переменной name_root	2
2		Ввод значения переменной name_root	3
3		Вызов метода input_name(name_root) по переменной name_root	4
4		Объявление переменной cl_base* temp_parent_name с указателем на текущий объект	5
5		Объявление переменной cl_base* temp_child_name со значением nullptr	6
6		Объявление переменных string name_parent и name_child	7
7		Ввод значений переменных name_parent и name_child	8
8	Значение name_parent = значение name_child		∅
			9
9	Значение свойства name	Создает новый объект класса cl_devider при	7

№	Предикат	Действия	№ перехода
	объекта temp_parent_name = значению name_parent	помощи соответствующего констрегтора передавая в качестве параметров указатель temp_parent_name и наименование name_child присваивая переменной temp_child_name значение указатель на этот созданный объект	
		Значение temp_parent_name = temp_child_name	10
10		Создает новый объект класса cl_devider при помощи соответствующего констрегтора передавая в качестве параметров указатель temp_parent_name и наименование name_child присваивая переменной temp_child_name значение указатель на этот созданный объект	7

3.12 Алгоритм конструктора класса cl_devider

Функционал: Присваивает полям объекта значения.

Параметры: нет.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса cl_devider

№	Предикат	Действия	№ перехода
1		Вызывает конструктор класса cl_base, с параметрами parent и name	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-15.

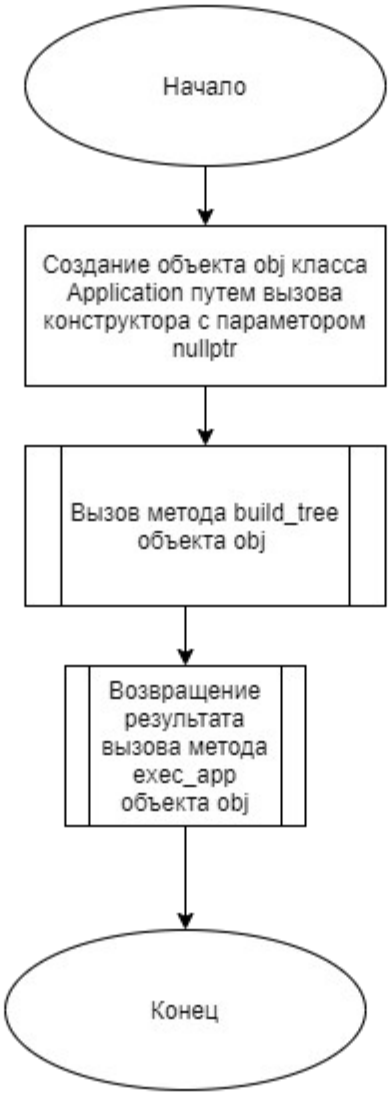

Рисунок 1 – Блок-схема алгоритма

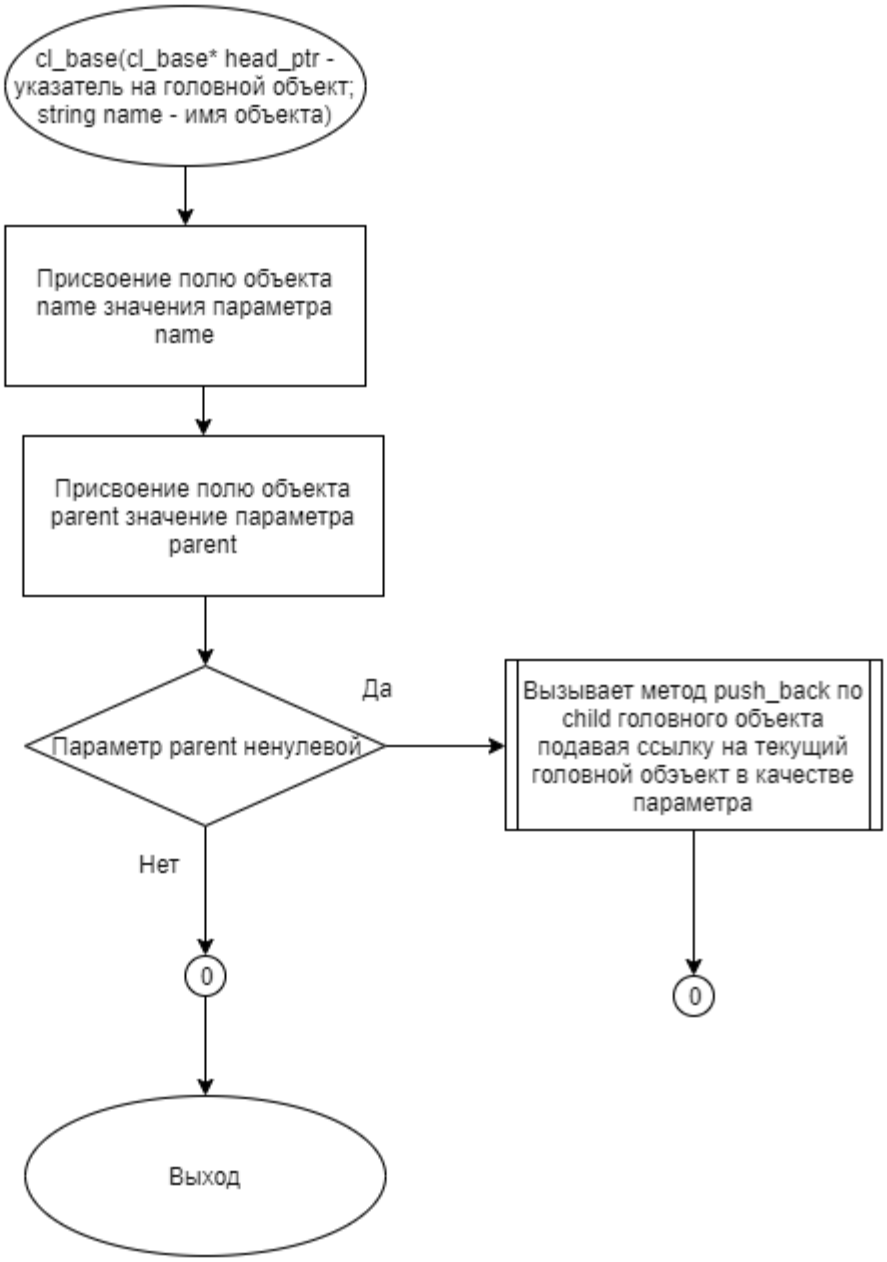

Рисунок 2 – Блок-схема алгоритма

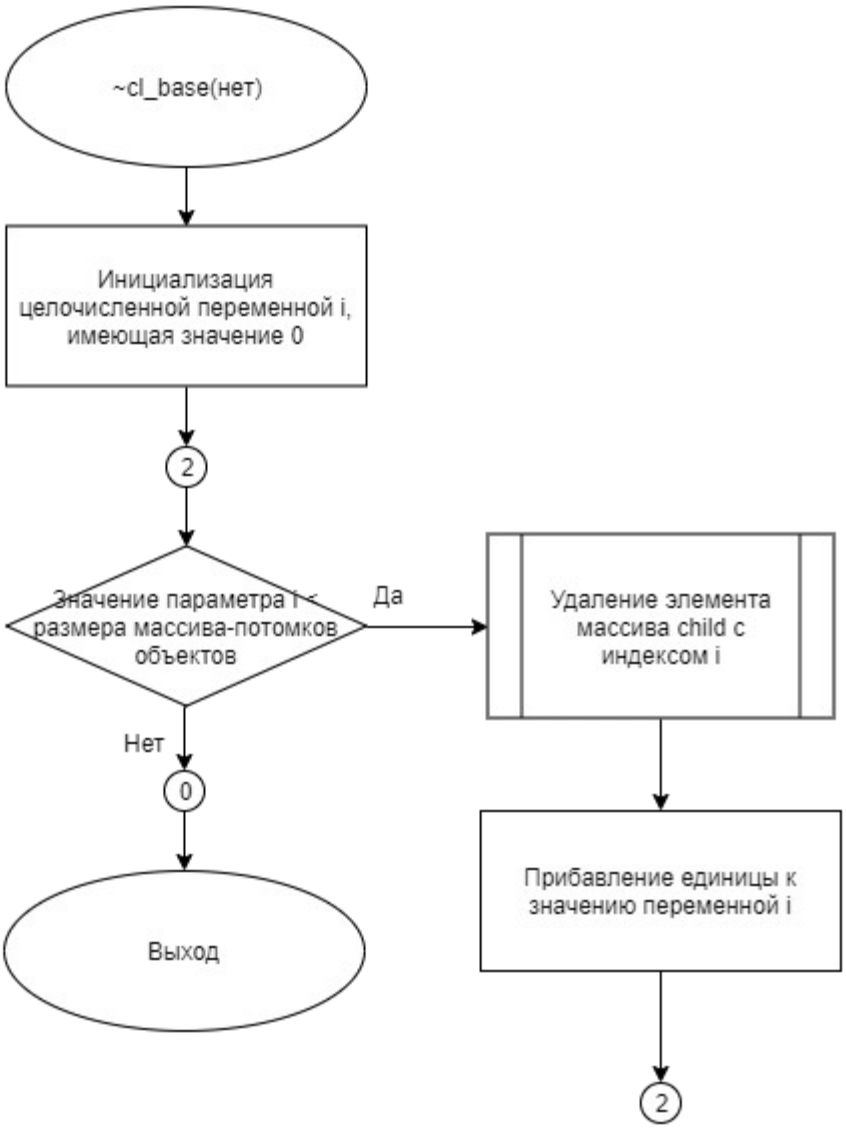

Рисунок 3 – Блок-схема алгоритма

Рисунок 4 – Блок-схема алгоритма

Рисунок 5 – Блок-схема алгоритма

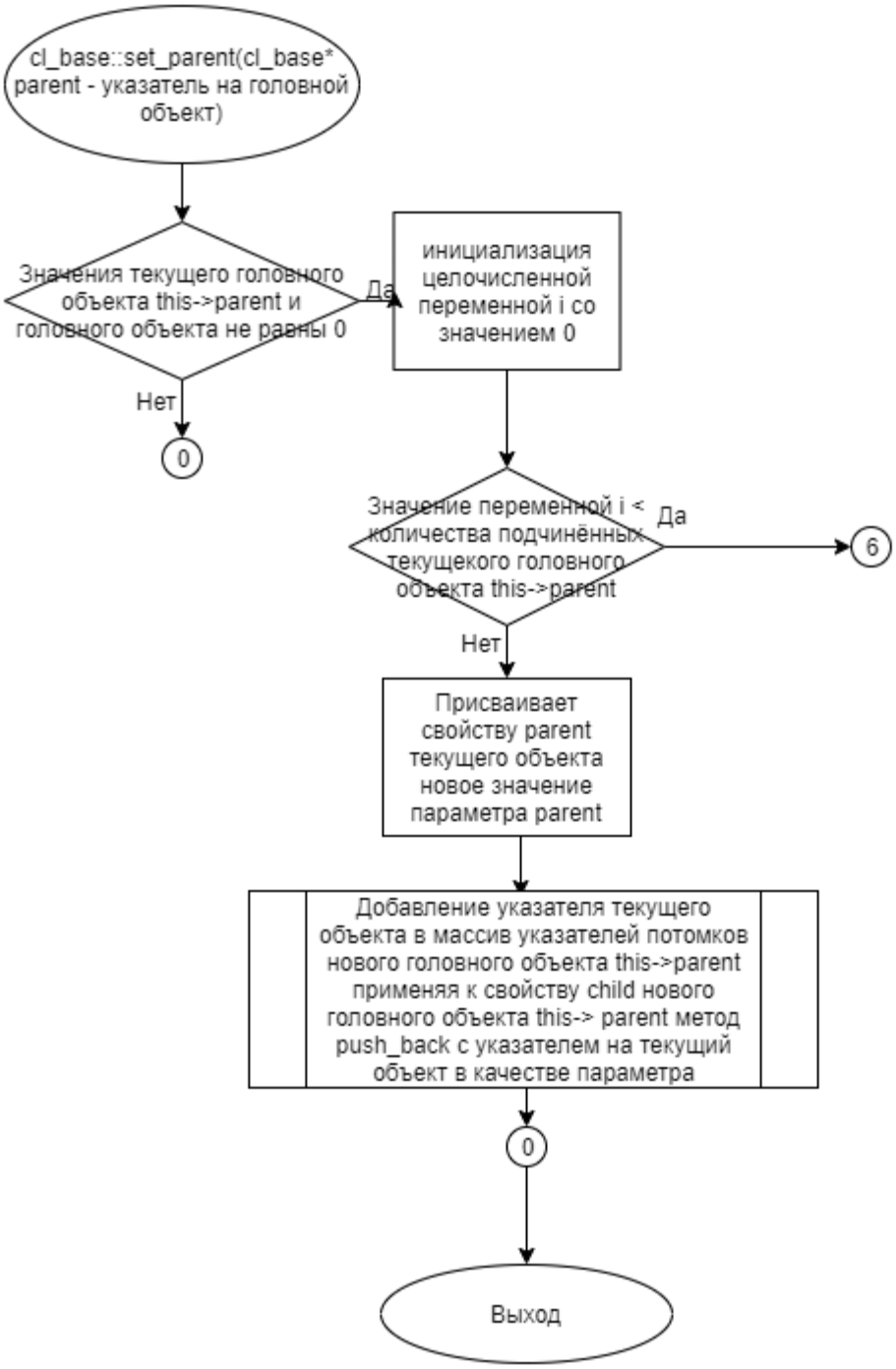

Рисунок 6 – Блок-схема алгоритма

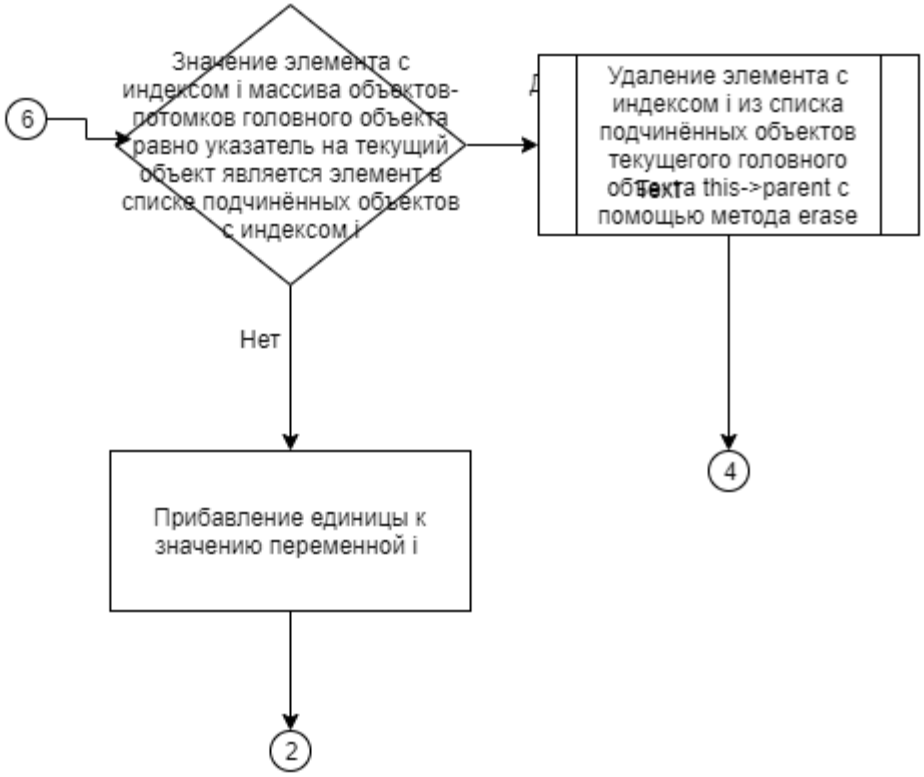

Рисунок 7 – Блок-схема алгоритма

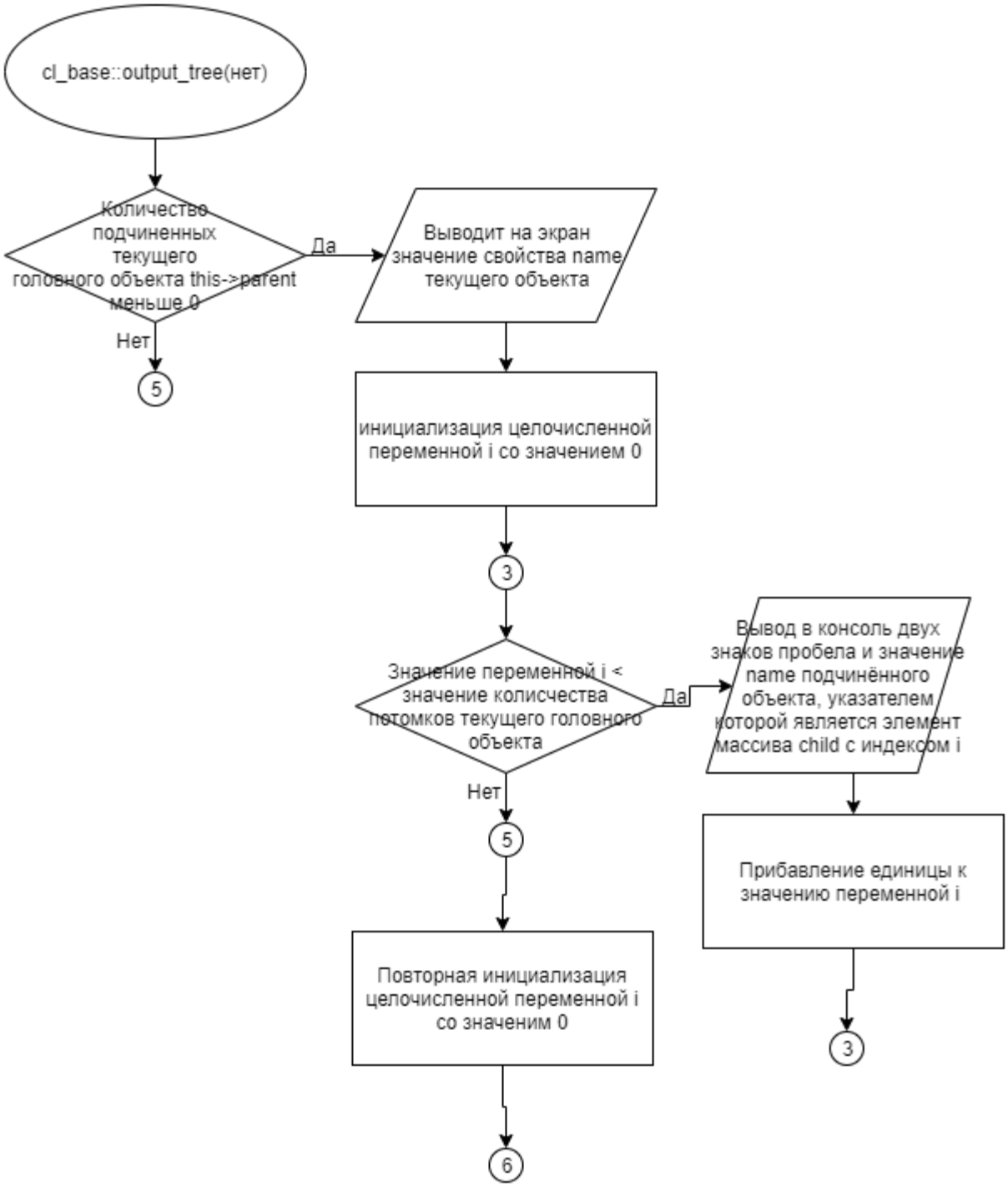

Рисунок 8 – Блок-схема алгоритма

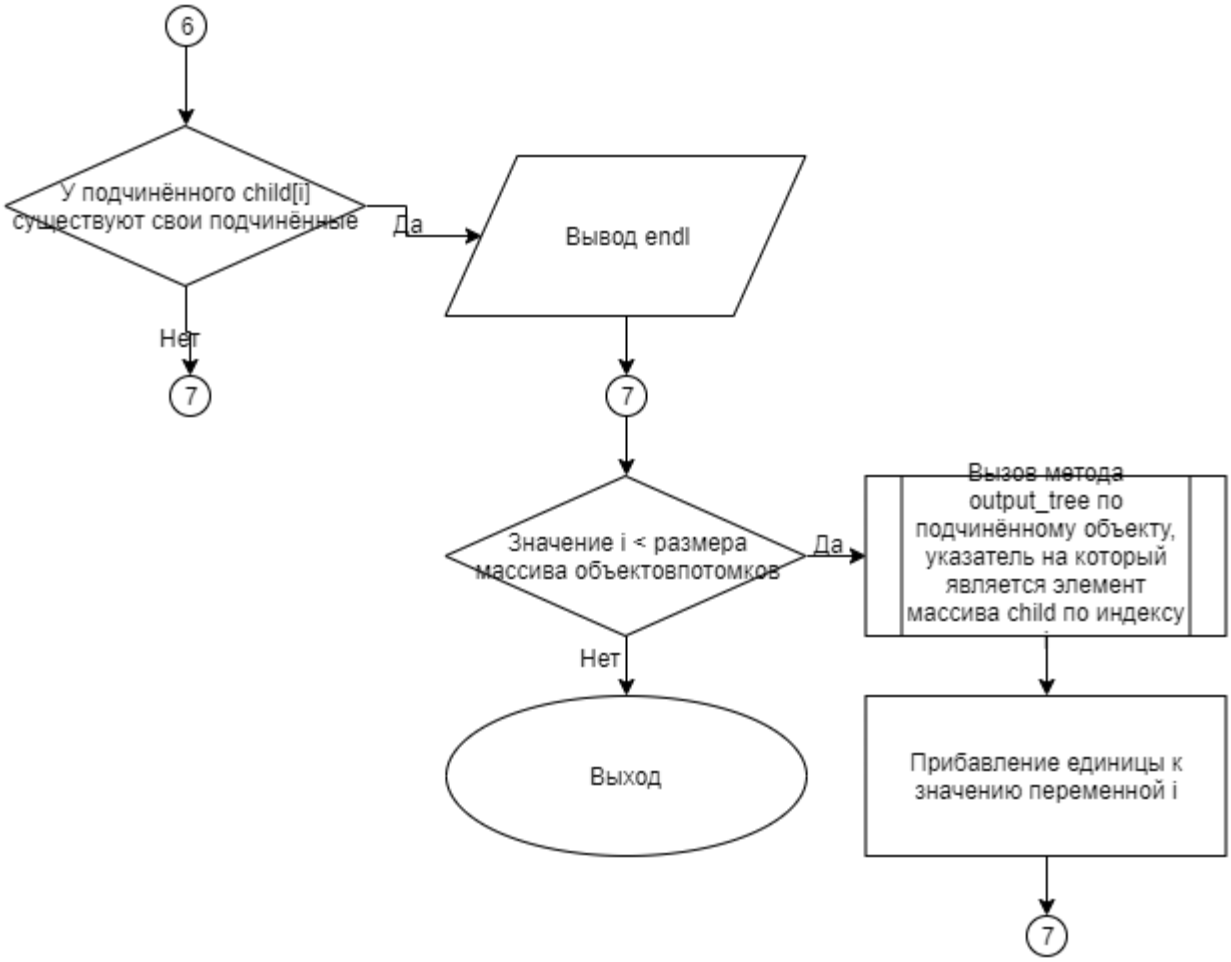

Рисунок 9 – Блок-схема алгоритма

Рисунок 10 – Блок-схема алгоритма

Рисунок 11 – Блок-схема алгоритма

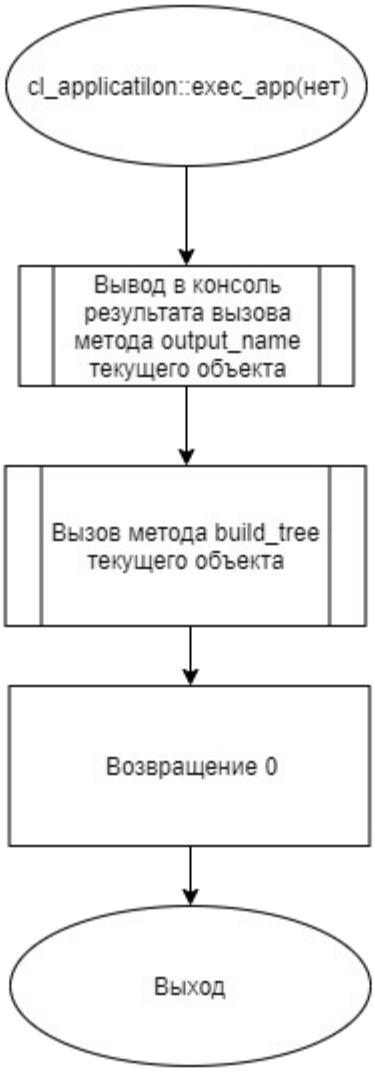

Рисунок 12 – Блок-схема алгоритма

Рисунок 13 – Блок-схема алгоритма

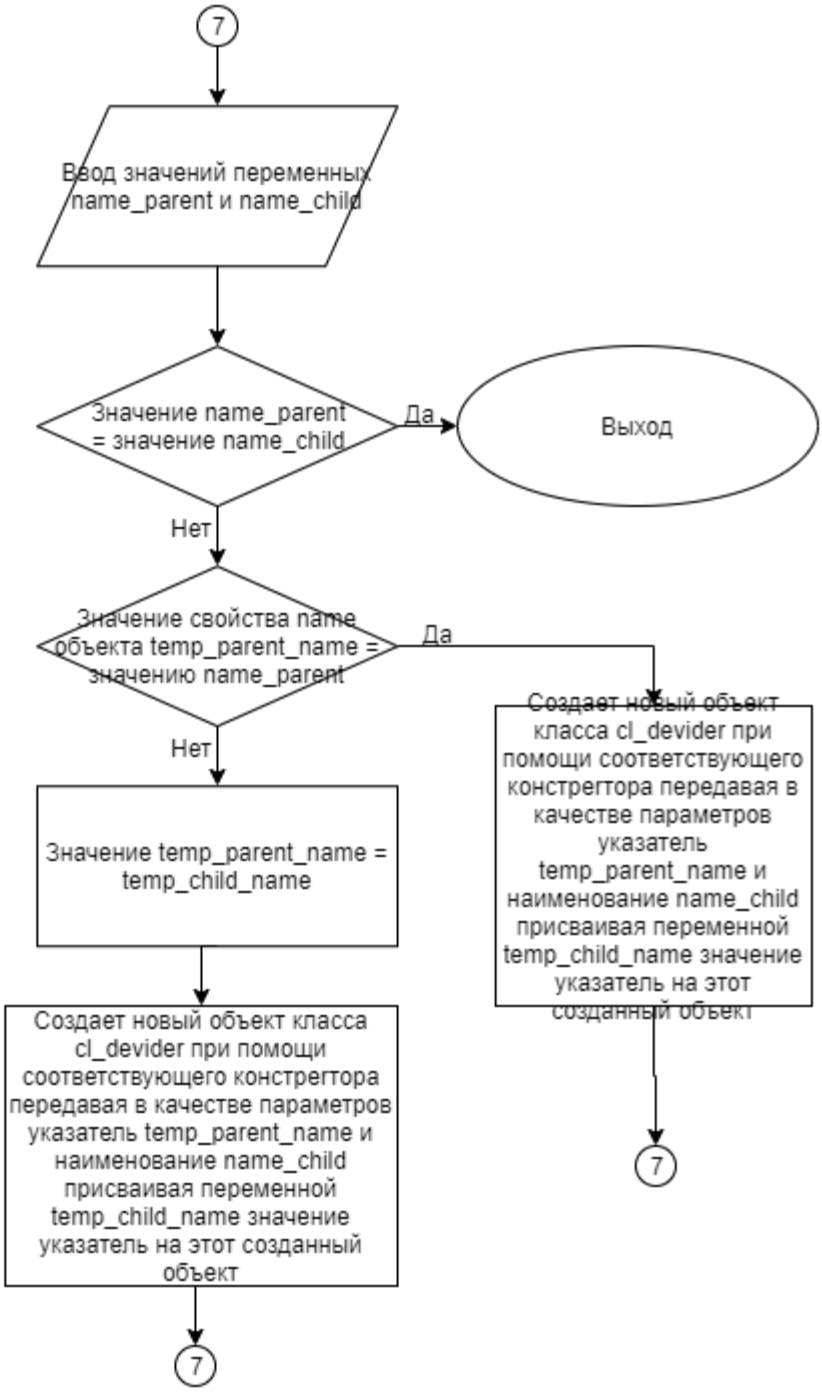

Рисунок 14 – Блок-схема алгоритма

Рисунок 15 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_application.cpp

Листинг 1 – cl_application.cpp

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

cl_application::cl_application(cl_base*  parent,  string  name):  cl_base(parent,
name){}
void cl_application::build_tree()
{
	string name_root, parent_name,child_name;
	cl_base* temp_parent_name, *temp_child_name;
	temp_parent_name = this;
	temp_child_name = nullptr;
	cin>>name_root;
	input_name(name_root);
	while(true)
	{
		cin>>parent_name>>child_name;
		if(parent_name== child_name) break;
		if (parent_name == temp_parent_name ->output_name() )
		{
			temp_child_name= new cl_devider(temp_parent_name, child_name);

		}
		else
		{
			temp_parent_name = temp_child_name;
			temp_child_name= new cl_devider(temp_parent_name, child_name);
		}
	}
}
int cl_application::exec_app()
{
	cout<< output_name () << endl;
	output_tree();
	return 0;
}
```

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H
#include <iostream>
#include <vector>
#include <string>
#include "cl_base.h"
#include "cl_devider.h"
class cl_application: public cl_base{

      public:
            cl_application(cl_base* parent, string name = " ");
            void build_tree();
            int exec_app();

};
#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include "cl_base.h"
using namespace std;
cl_base::cl_base(cl_base* parent, string name)
{
      this->name=name;
      this->parent=parent;
      if (parent!=nullptr) parent->child.push_back(this);
}
cl_base::~cl_base()
{
      for (int i=0; i < this->child.size();i++)
            delete this->child[i];
}
void cl_base::input_name(string name)
{
      this->name = name;
}
string cl_base::output_name()
{
      return name;
}
void cl_base::set_parent(cl_base* parent)
{
```

```
	if(this->parent!=nullptr && parent!=nullptr)
	{
		for(int i=0; i<this->parent->child.size(); i++)
		{
			if(this==this->parent->child[i])
			{
				this->parent->child.erase(child.begin()+i);
			}
		}
		this->parent= parent;
		parent->child.push_back(this);
	}
}
void cl_base::output_tree()
{
	if(this->child.size()>0)
	{
		cout << name;
		for(int i=0; i<child.size(); i++)
		{
			cout << "  " << child[i]->name;
		}
	}
	for(int i=0; i < child.size(); i++) {
		if(child[i]->child.size()>0) {
			cout << endl;
		}
		child[i]->output_tree();
	}
}

cl_base* cl_base::output_parent()
{
	return parent;
}
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H
#include <iostream>
#include <string>
#include <vector>
#include <stdlib.h>
#include <stdio.h>
using namespace std;
class cl_base
{
	string name;
	cl_base* parent;
	vector <cl_base*> child;
```

```
      public:
            cl_base(cl_base* parent, string name=" ");
            ~cl_base();
            void input_name(string name);
            string output_name();
            void output_tree();
            void set_parent(cl_base* parent);
            cl_base* output_parent();

};

#endif
```

5.5 Файл cl_devider.cpp

Листинг 5 – cl_devider.cpp

```
#include "cl_devider.h"
cl_devider::cl_devider(cl_base* parent, string name): cl_base(parent, name) {}
```

5.6 Файл cl_devider.h

Листинг 6 – cl_devider.h

```
#ifndef CL_Devider_H
#define CL_Devider_H
#include "cl_base.h"
using namespace std;
class cl_devider : public cl_base
{
public:
cl_devider (cl_base* parent, string name);
};

#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <vector>
#include <string>
```

```
#include "cl_application.h"
int main()
{
	cl_application obj(nullptr);
	obj.build_tree();
	return obj.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_5 Object_6 Object_7 Object_7	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5 Object_5 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5 Object_5 Object_6
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).