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;
- € 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.

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

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

Метод решения

Для решения поставленной задачи используются функции ввода вывода cin/cout, условный оператор if/else циклы for/while, а так же библиотеки string/vector;

клысс: cl_base и его наследник cl_application

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	описание	номер	комментарий
1	cl_base			базовый класс. Содержит основные поля и методы		
		cl_application	public		2	
2	cl_application			класс приложения(нео		

				бходим для запуска и работы самой программы)		

класс cl_base содержит поля:

1. 1. string object_name - имя объектка
2. 2. cl_base* parent - указатель на родительский объект (для текущего объектка)
3. 3. static cl_base* root - корневой (технический, не доступный пользователю) объект
4. 4. vector <cl_base*> children - вектор, хранящий в себе сам объект и его прямых наследников

а так же методы:

1. cl_base(string object_name, cl_base* parent) - параметризированный конструктор
2. void set_name(string name) - метод сохранения имени
3. void set_parent(cl_base* parent) - метод сохранения родительского объекта
4. string get_name() - метод получения имени
5. cl_base* get_object_by)name(string name) - метод получения ссылки на объект по его имени
6. void print_tree() - метод вывода в консоль дерева объектов

класс cl_application содержит методы:

1. void bild_tree_objects() - метод постройки дерева объектов (базового класса)
2. int exe_app() - метод вывода головного объекта и его дерева наследников (за счет метода базового класса print_tree)

Описание алгоритма

Функция: main

Функционал: точка входа

Параметры: нет

Возвращаемое значение: 0

№	Предикат	Действия	№ перехода	Комментарий

1		создает объект класса cl_application	2	
2		вызываает метод bild_tree_objects	3	
3		вызываает метод exec_app	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: cl_application(cl_base* parent) - конструктор

Функционал: заполняет технический объект root

Параметры: cl_base* parent - указатель на родительский объкт

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		cl_base::root->set_parent(parent);	2	
2		cl_base::root->set_name("root");	3	
3		cl_base::root->children.push_back(root);	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: bild_tree_objicts()

Функционал: строит дерево объектов

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		ввод имени корневого объекта	2	
2		создание коренвого объекта	3	
3		ввод имени родителя, ввод имени ребенка	4	
4	(name1 == name2)	выход	∅	
		переход	5	
5		получение ссылки на объект родителя	6	
6		создание дочернего объекта с передачей в него имени и ссылки на родительский объект	3	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app()

Функционал: выводит дерево объектов в соответствии с тз

Параметры: нет

Возвращаемое значение: 0

№	Предикат	Действия	№ перехода	Комментарий
1		вывод имени корневого объекта	2	
2		вызов метода базового класса print_tree	3	
3		выход из метода, возврат 0	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base()

Функционал: не параметризированный конструктор

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		он есть	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base(string object_name, cl_base* parent)

Функционал: создает объект и схраняет имя и указатель на родителя

Параметры: имя объекта и указатель на объект родитель

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет имя объекта	2	
2	(parent == nullptr)	сохранить как родительский объект root в родительский vector сохранить текущий объект	3	

		сохранить в родительский объект тот, что был передан параметром в родительский vector сохранить текущий объект	3	
3		в vector сохранить текущий объект	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_name(string name)

Функционал: сохраняет имя

Параметры: имя объекта

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет имя объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_parent(cl_base* parent)

Функционал: сохраняте указатель на родительский объект

Параметры: указатель на родительский объект

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняте указатель на родительский объект	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_name()

Функционал: возвращает имя объекта

Параметры: нет

Возвращаемое значение: string (имя объекта)

№	Предикат	Действия	№ перехода	Комментарий
1		возвращает имя объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_object_by_name(string name)

Функционал: возвращает указатель на объект по его имени

Параметры: имя объекта

Возвращаемое значение: указатель на объект

№	Предикат	Действия	№ перехода	Комментарий
1	(i < children.size())	переход	2	
		переход	3	
2	(children[i]->get_name() == nsme)	возврат children[i] выход	∅	
		переход	1	
3	(i < children.size())	вызов get_object_by_name(string name) от имени дочернего объекта	4	
		выход	∅	
4	((children[i]->get_object_by_name(string name))->get_name() == name)	выход	∅	
		переход	3	

Класс объекта: cl_base

Модификатор доступа: public

Метод: print_tree

Функционал: выводит дерево объектов

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1	(i < children.size())	вывод children[i]	1	
		переход	2	
2	(i < children.size())	переход	3	

		выход	∅	
3	(children[i]->children.size() > 1)	вызов print_tree()	2	
		переход	2	

Класс объекта: cl_base

Модификатор доступа: public

Метод: ~cl_base()

Функционал: деструктор

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		он есть	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_parent()

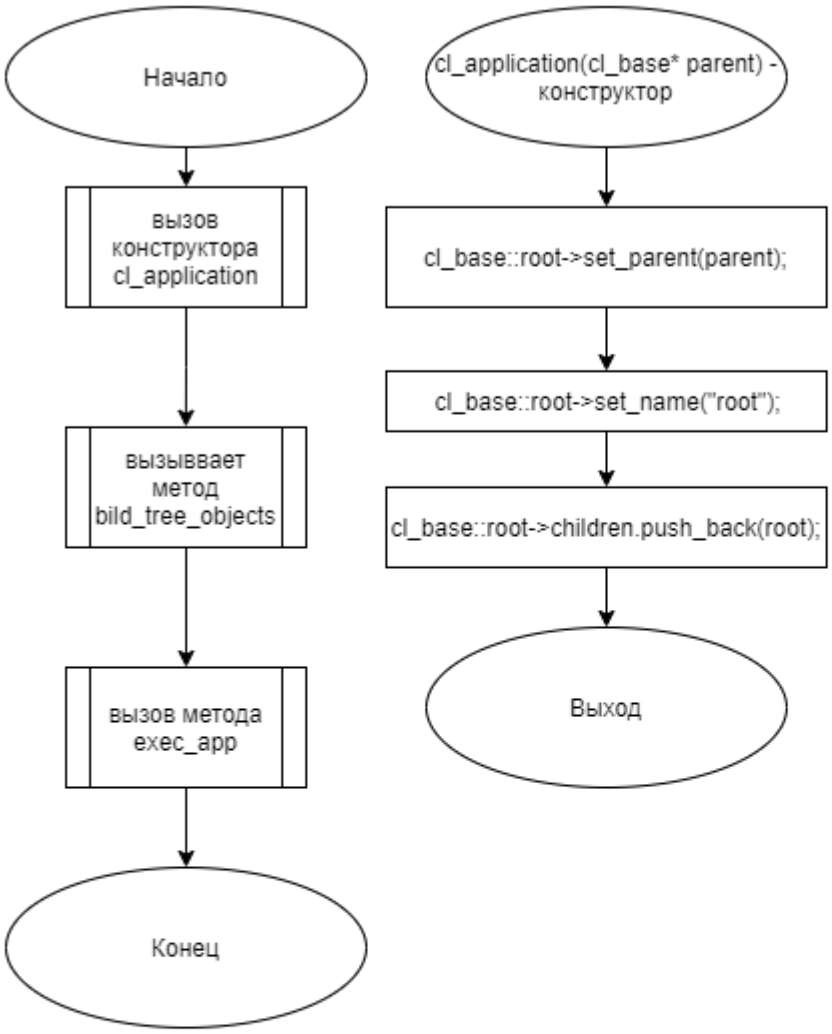
Функционал: возвращает указатель на родительский объект

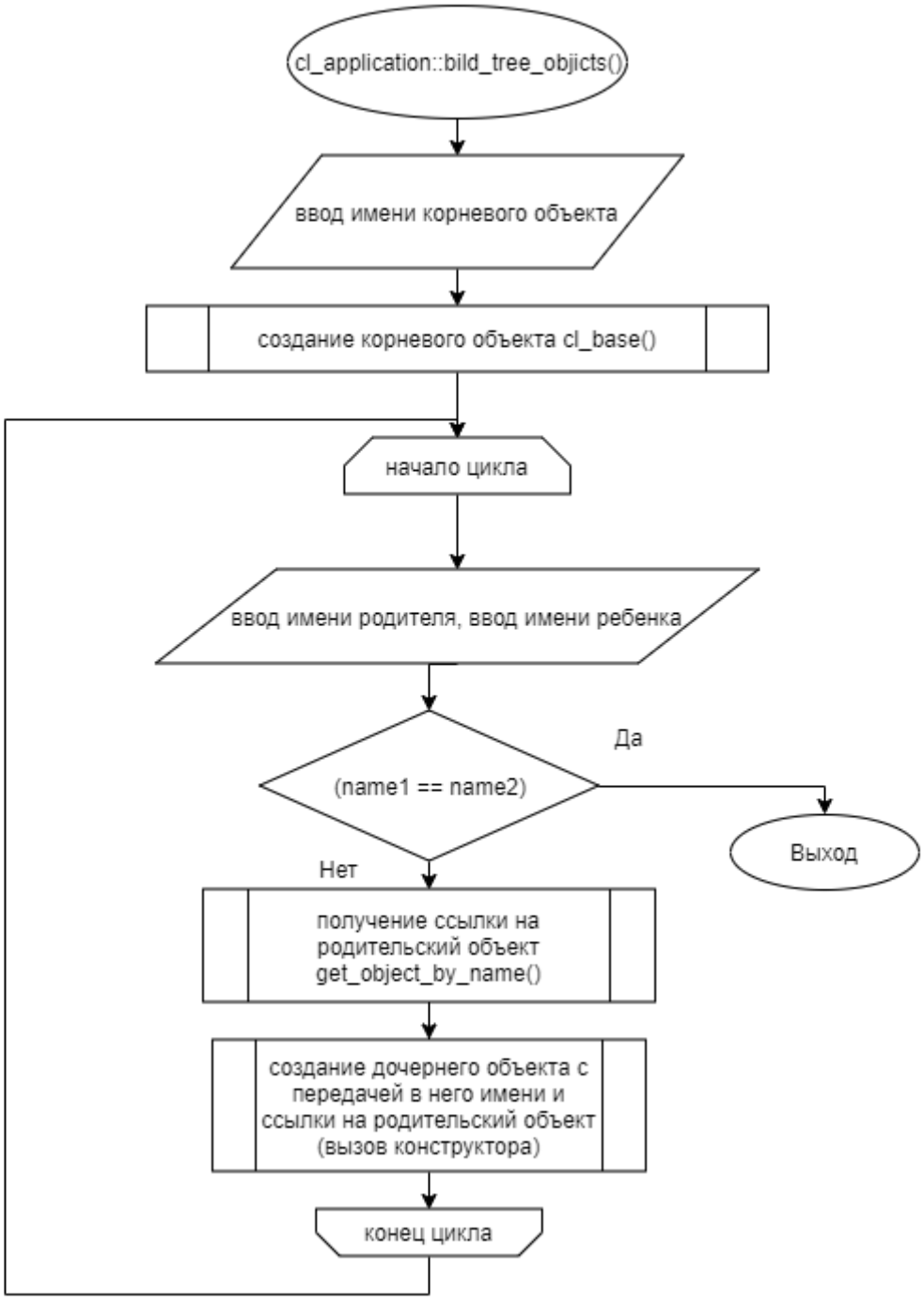
Параметры: нет

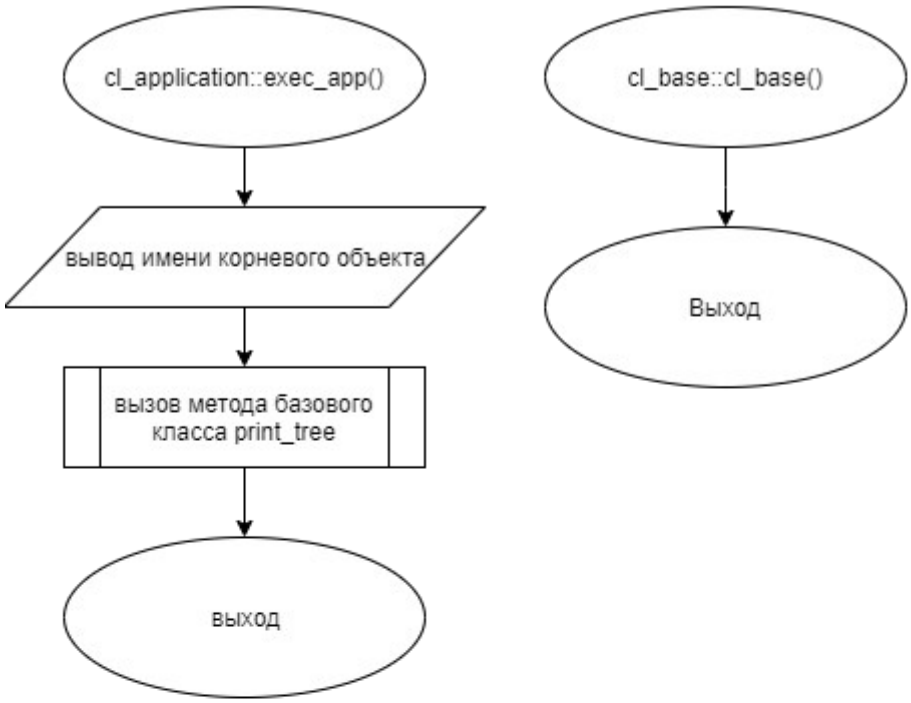
Возвращаемое значение: *parent (указатель на родительский объект)

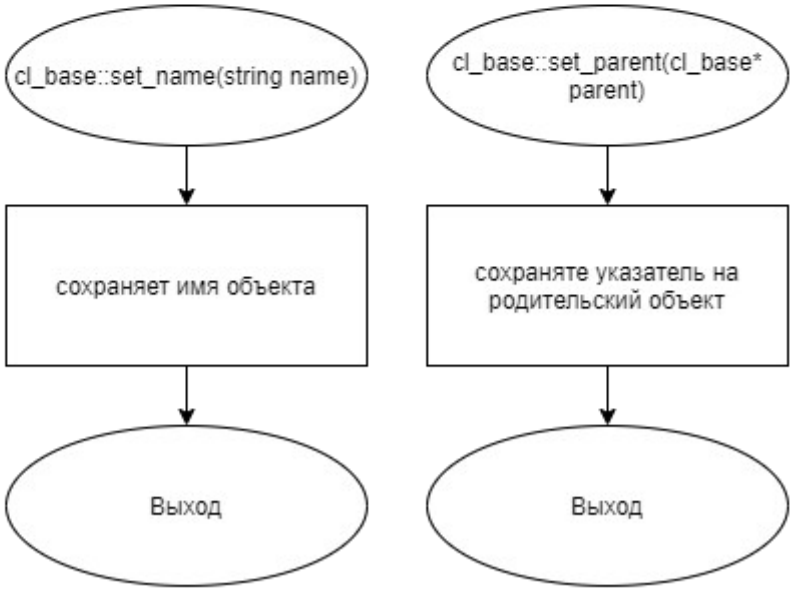
№	Предикат	Действия	№ перехода	Комментарий
1		возвращает указатель на родительский объект	∅	

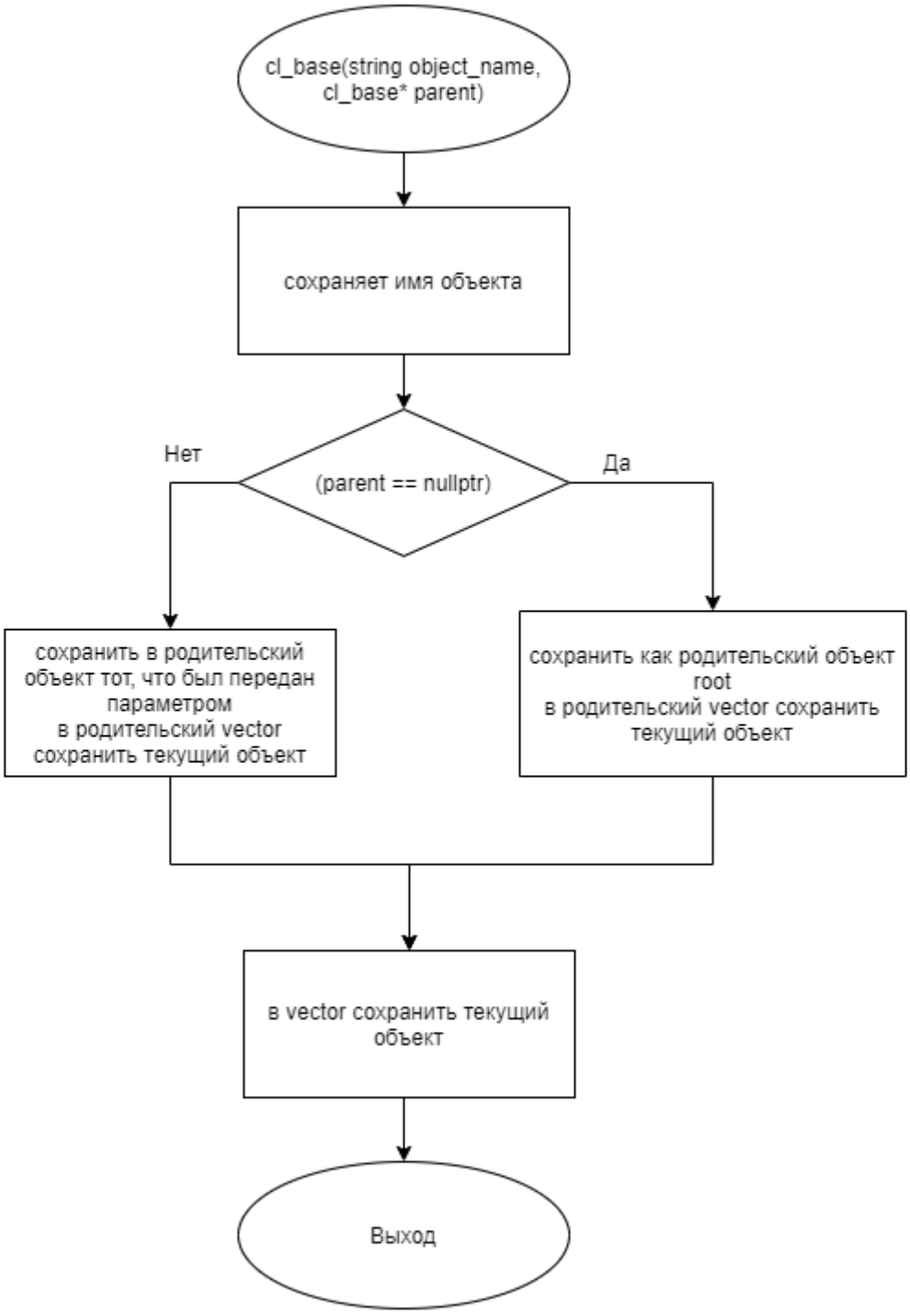
Блок-схема алгоритма

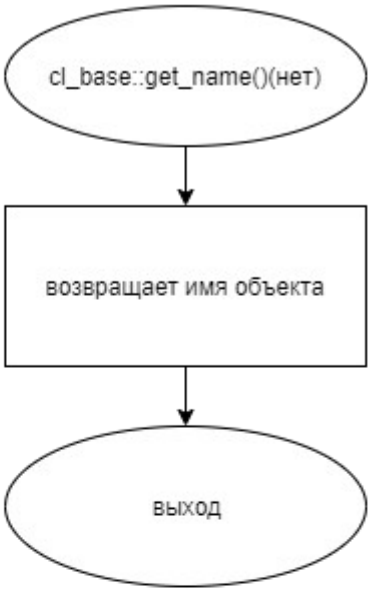

Начало
вызов конструктора cl_application
вызываает метод bild_tree_objects
вызов метода exec_app
Конец
cl_application(cl_base* parent) - конструктор
cl_base::root->set_parent(parent);
cl_base::root->set_name("root");
cl_base::root->children.push_back(root);
Выход

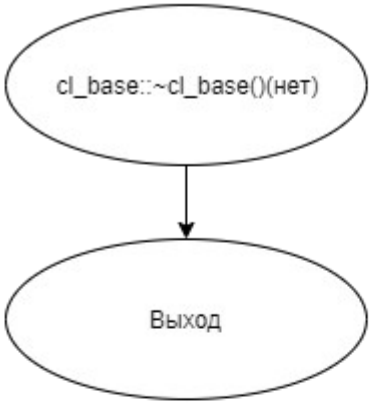

cl_application::bild_tree_objicts()
ввод имени корневого объекта
создание корневого объекта cl_base()
начало цикла
ввод имени родителя, ввод имени ребенка
(name1 == name2)
Да
Выход
Нет
получение ссылки на родительский объект get_object_by_name()
создание дочернего объекта с передачей в него имени и ссылки на родительский объект (вызов конструктора)
конец цикла

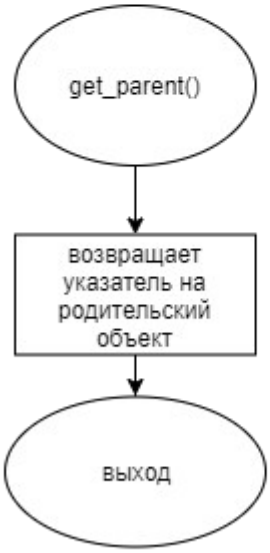

cl_application::exec_app()
вывод имени корневого объекта
вызов метода базового класса print_tree
выход
cl_base::cl_base()
Выход

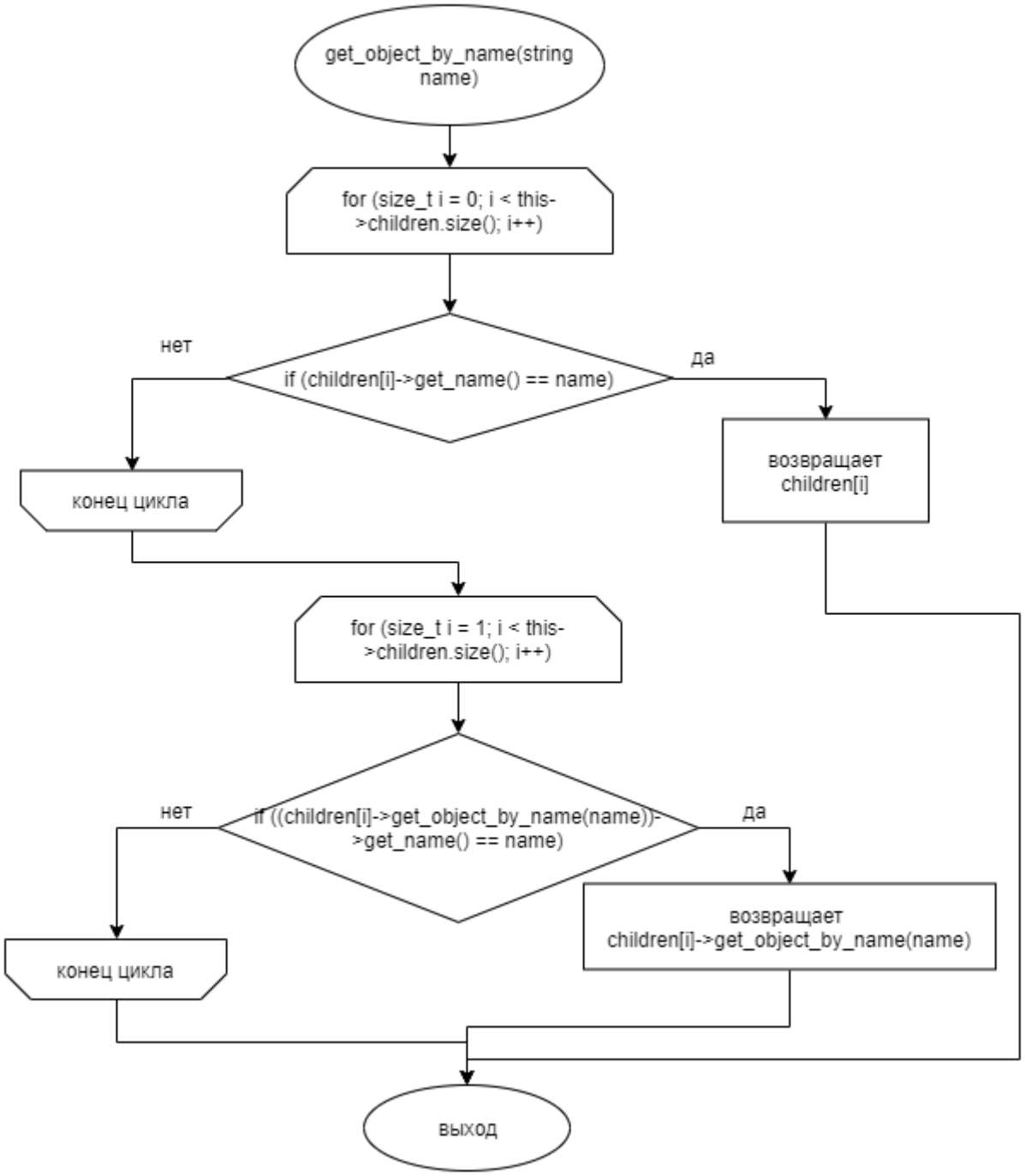

cl_base::set_name(string name)
сохраняет имя объекта
Выход
cl_base::set_parent(cl_base* parent)
сохраняте указатель на родительский объект
Выход

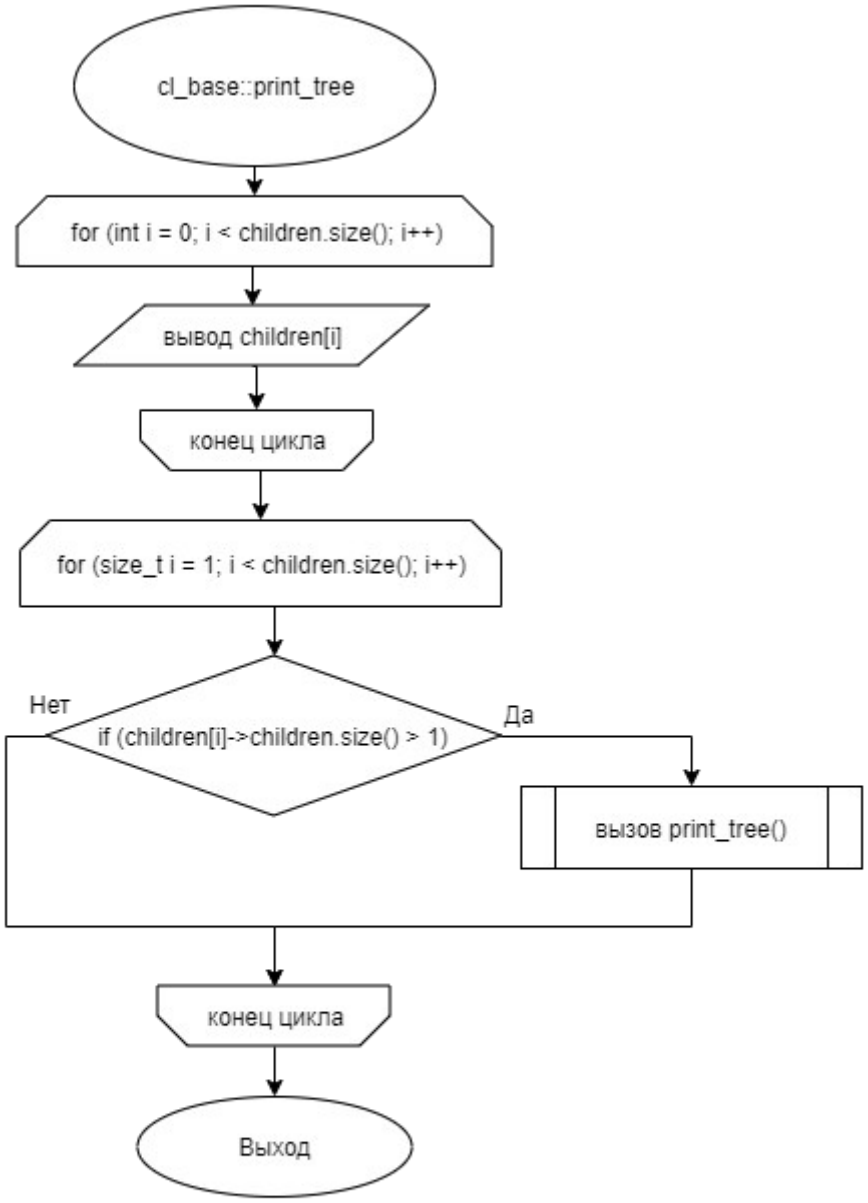

cl_base(string object_name,
cl_base* parent)
сохраняет имя объекта
(parent == nullptr)
Нет
Да
сохранить в родительский объект тот, что был передан параметром
в родительский vector сохранить текущий объект
сохранить как родительский объект root
в родительский vector сохранить текущий объект
в vector сохранить текущий объект
Выход


cl_base::get_name()(нет)
возвращает имя объекта
выход


cl_base::~cl_base()(нет)
Выход


get_parent()
возвращает указатель на родительский объект
выход


get_object_by_name(string name)
for (size_t i = 0; i < this->children.size(); i++)
нет
if (children[i]->get_name() == name)
да
конец цикла
возвращает children[i]
for (size_t i = 1; i < this->children.size(); i++)
нет
if ((children[i]->get_object_by_name(name))->get_name() == name)
да
конец цикла
возвращает children[i]->get_object_by_name(name)
выход



Код программы

Файл cl_application.cpp

```
#include "cl_application.h"
cl_application::cl_application(cl_base* parent = nullptr)
{
        cl_base::root->set_parent(parent);
        cl_base::root->set_name("root");
        cl_base::root->children.push_back(root);
}
void cl_application::bild_tree_objects()
{
        string name1, name2;
        cin >> name1;
        cl_base* child = new cl_base(name1, nullptr);
        while (true)
        {
                cin >> name1 >> name2;
                if (name1 == name2)
                {
                        return;
                }
                cl_base* child2 = new cl_base(name2, root-
>get_object_by_name(name1));
                child = child2;
        }
}
int cl_application::exec_app()
{
        cout << root->children[1]->get_name();
        root->children[1]->print_tree();
        return 0;
}
```

Файл cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H

#include "cl_base.h"
#include <string>
using namespace std;

class cl_application : public cl_base
{
public:
        cl_application(cl_base* parent);
        void bild_tree_objects();
        int exec_app();
};


#endif
```

Файл cl_base.cpp

```
#include "cl_base.h"
#include <string>
cl_base* cl_base::root = new cl_base();
cl_base::cl_base()
{
	parent = nullptr;
}
cl_base::cl_base(string object_name, cl_base* parent)
{
	this->object_name = object_name;
	if (parent == nullptr)
	{
		set_parent(root);
		(this->parent)->children.push_back(this);
	}
	else
	{
		this->parent = parent;
		parent->children.push_back(this);
	}
	children.push_back(this);
	index = (this->parent)->children.size() - 1;
}
void cl_base::set_name(string name)
{
	this->object_name = name;
}
void cl_base::set_parent(cl_base* parent)
{
	this->parent = parent;
}
///
cl_base* cl_base::get_parent()
{
	return parent;
}
///
string cl_base::get_name()
{
	return object_name;
}
cl_base* cl_base::get_object_by_name(string name)
{
	cl_base* val = nullptr;
	bool chek = false;
	for (size_t i = 0; i < this->children.size(); i++)
	{
		val = children[i];
```

```
                if (children[i]->get_name() == name)
                {
                        chek = true;
                        return children[i];
                }

                /*
                for (size_t i = 1; i < children.size(); i++)
                {
                        val = (children[i]->get_object_by_name(name));
                        if ((children[i]->get_object_by_name(name))-
>get_name() == name)
                        {
                                return (children[i]-
>get_object_by_name(name));
                        }
                }
                */
        }

        for (size_t i = 1; i < children.size(); i++)
        {
                val = (children[i]->get_object_by_name(name));
                if ((children[i]->get_object_by_name(name))->get_name() ==
name)
                {
                        return (children[i]->get_object_by_name(name));
                }
        }

        return val;
}
void cl_base::print_tree()
{
        for (int i = 0; i < children.size(); i++)
        {
                if (i == 0)
                        cout << endl;
                cout << children[i]->get_name();
                if(i+1 < children.size())
                        cout << "  ";
        }
        for (size_t i = 1; i < children.size(); i++)
        {
                if (children[i]->children.size() > 1)
                {
                        children[i]->print_tree();
                }
        }
}
cl_base::~cl_base()
{
}
```

Файл cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H

#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class cl_base
{
        string object_name = "";
        cl_base* parent;
        int index = 0;
public:
        int iterator;
        static cl_base* root;
        vector <cl_base*> children;
        cl_base();
        cl_base(string object_name, cl_base* parent);
        void set_name(string name);
        void set_parent(cl_base* parent);
        cl_base* get_parent();
        string get_name();
        cl_base* get_object_by_name(string name);
        void print_tree();
        ~cl_base();
};

#endif
```

Файл main.cpp

```
#include "cl_application.h"
int main()
{
        setlocale(LC_ALL, "ru");
        cl_application ob_cl_application(nullptr);
        ob_cl_application.bild_tree_objects();
        return ob_cl_application.exec_app();
}
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
I I tired_to_do tired_to_do it_normal fail fail	I I tired_to_do tired_to_do it_normal	I I tired_to_do tired_to_do it_normal
1 1 2 1 3 1 4 2 2.1 2 2.2 3 3.1 3 3.2 4 4.1 4 4.2 4.2 4.2.1 5 5	1 1 2 3 4 2 2.1 2.2 3 3.1 3.2 4 4.1 4.2 4.2 4.2.1	1 1 2 3 4 2 2.1 2.2 3 3.1 3.2 4 4.1 4.2 4.2 4.2.1