Постановка задачи

Проверка готовности объектов к работе

Фрагмент методического указания.
Создание объектов и построение исходного иерархического дерева объектов.
Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность).
Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно.

Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main). Исходное состояние корневого объекта соответствует его функционированию.
Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии.
Структура данных в строке:
«Наименование головного объекта»«Наименование очередного объекта»«Номер класса принадлежности очередного объекта»«Номер исходного состояния очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно « endtree » (в данной строке ввода больше ничего не указывается).

Готовность объекта характеризуется значением его состояния. Значение состояния - целое число.
Определены правила для значения состояния:
0 – объект выключен;
Отрицательное – объект включен, но не функционирует, обнаружена неисправность. Значение классифицирует характер неисправности.
Положительное – объект включен, функционирует в штатном режиме. Значение определяет текущее состояние объекта.

Подчиненные объекты располагаются слева на право относительно головного, согласно их следованию в исходных данных. Исходные данные подготовлены таким образом, что любой головной объект предварительно добавлен в качестве подчиненного.
Подразумевается, что все объекты имеют уникальные имена.
Для организации исходя из входных данных создания экземпляров объектов и формирования иерархического дерева, необходимо:

1. В базовом классе реализовать метод поиска объекта на дереве объектов по его наименованию и возврата указателя на него. Если объект не найден, то вернуть нулевой указатель.
2. В корневом объекте (объект приложения) реализовать метод чтения исходных данных, создания объектов и построения исходного дерева иерархии.

Пример

Ввод

```
app_root

app_root object_1 3 1

app_root object_2 2 1

object_2 object_4 3 -1

object_2 object_5 3 1

app_root object_3 3 1

object_2 object_6 2 1

object_1 object_7 2 1

endtree
```

Построенное дерево

```
app_root

    object_1

    object_7

    object_2

    object_4
```

```
object_5


object_6


object_3
```

Вывод списка готовности объектов

```
The object app_root is ready

The object object_1 is ready

The object object_7 is ready

The object object_2 is ready

The object object_4 is not ready

The object object_5 is ready

The object object_6 is ready

The object object_3 is ready
```

Постановка задачи

Все сложные электронные, технические средства разного назначения в момент включения выполняют опрос готовности к работе составных элементов, индицируя соответствующую информацию на табло, панели или иным образом.
Построить модель иерархической системы. Реализовать задачу опроса готовности каждого объекта из ее состава и вывести соответствующее сообщение на консоль.
Объект считается готовым к работе:

1. Создан и размешен в составе системы (на дереве иерархии объектов) согласно схеме архитектуры;
2. Имеет свое уникальное наименование;

3. Свойство, определяющее его готовность к работе, имеет целочисленное положительное значение.

В результате решения задачи опроса готовности объектов, относительно каждого объекта системы на консоль надо вывести соответствующую информацию:
Если свойство определяющее готовность объекта имеет положительное значение:
The object «наименование объекта» is ready
иначе
The object « наименованиеобъекта » is not ready
Система содержит объекты трех классов, не считая корневого. Номера классов: 2,3,4.

Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в фрагменте методического указания.

Описание выходных данных

В первой строке вывести
Test result

Далее, построчно, согласно следованию объектов на дереве иерархии слева на право и сверху вниз, относительно каждого объекта в зависимости от состояния готовности выводиться,
если объект готов к работе:
The object «наименование объекта» is ready

Если не готов, то:
The object «наименование объекта» is not ready

Метод решения

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

классы: cl_base, cl_application, cl2, cl3, cl4

№	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	
		cl2	public		3	
		cl3	public		4	
		cl4	public		5	
2	cl_application			класс приложения (необходим для запуска и работы самой программы)		
3	cl2			необходим по условия задания		
4	cl3			необходим по условия задания		
5	cl4			необходим по условия задания		

класс cl_base содержит поля:

1. string object_name - имя объекта
2. cl_base* parent - указатель на родительский объект (для текущего)
3. static sl_base* root - корневой (не доступный пользователю) объект
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 указателя на объект-родитель
4. string get_name() - метод получения имени объекта
5. cl_base* get_object_by_name(string name) - метод получения указателя на объект по его имени

6. void print_tree() - метод вывода в консоль объектов

класс cl_application содержит методы:

1. void bild_tree_bjects() - метод постройки дерева объектоа
2. int exe_app() - метод итогового вывода (по условию задания)

класс cl2, cl3, cl4 имеют только констукторы и необходимы исключительно по тз

Описание алгоритма

Функция: main

Функционал: точка входа

Параметры: нет

Возвращаемое значение: 0

№	Предикат	Действия	№ перехода	Комментарий
1		создание объекта класса cl_application (вызов конструктора класса cl_application)	2	
2		вызов метода класса cl_application bild_tree_objects()	3	
3		вызов метода класса cl_application exec_app	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: cl_application (конструктор)

Функционал: конструктор

Параметры: cl_base* parent

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_objects

Функционал: строит дерево объектов

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1	(true)	ввод имени объекта	2	
		иначе (не бывает)	∅	
2	(name1 == "endtree")	return	∅	
		переход	3	
3		ввод второго имени, номера класса, показателя готовности	4	
4		создание нового объекта (одного из классов cl2/cl3/cl4 в зависимости от введенных пользователем данных) с передачей параметрами имени указателя на родительский объект и флага готовности	1	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app

Функционал: выводит последовательность объектов

Параметры: нет

Возвращаемое значение: 0

№	Предикат	Действия	№ перехода	Комментарий
1		вывод "Test result"	2	
2		вызов метода сласса cl_base print_tree	3	
3		завершение метода, возврат 0	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base (конструктор)

Функционал: конструктор

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		выход	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base (параметризированны конструктор)

Функционал: конструктор

Параметры: string object_name, cl_base* parent, bool ready

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет флаг готовности объекта	2	
2		сохраняет имя объекта	3	
3		сохраняет указатель на родительский объект	4	
4	(parent != nullptr)	сохраняет в родительский vector указатель на себя	5	
		ничего	5	
5		сохраняет указатель на себя в свой vector	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_name

Функционал: сохраняет имя

Параметры: string name

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет имя объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_parent

Функционал: сохраняет указатель на родительский объект

Параметры: cl_base* parent

Возвращаемое значение: нет

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

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_name

Функционал: возвращает имя объекта

Параметры: нет

Возвращаемое значение: string

№	Предикат	Действия	№ перехода	Комментарий
1		возвращает имя объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_object_by_name

Функционал: возвращает указатель на объект по переданному имени

Параметры: string name

Возвращаемое значение: cl_base*

№	Предикат	Действия	№ перехода	Комментарий
1	(i<children.size())	проверка условия (if)	2	

		переход	3	
2	(children[i]->get_name() == name)	возврвт 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 последовательность объектов

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		выводит "The object " имя объекта	2	
2	(this->ready)	выводит " is ready"	3	
		выводит " is not ready"	3	
3	(i <children.size())	вызывает print_tree() от имени наследника i	3	
		выход	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: ~cl_base (деструктор)

Функционал: деструктор

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		выход	∅	

Класс объекта: cl2

Модификатор доступа: public

Метод: cl2

Функционал: вызывает конструктр базового класса

Параметры: string object_name, cl_base* parent, bool ready

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		вызывает конструктор базового класса	∅	

Класс объекта: cl3

Модификатор доступа: public

Метод: cl3

Функционал: вызывает конструктр базового класса

Параметры: string object_name, cl_base* parent, bool ready

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		вызывает конструктр базового класса	∅	

Класс объекта: cl4

Модификатор доступа: public

Метод: cl4

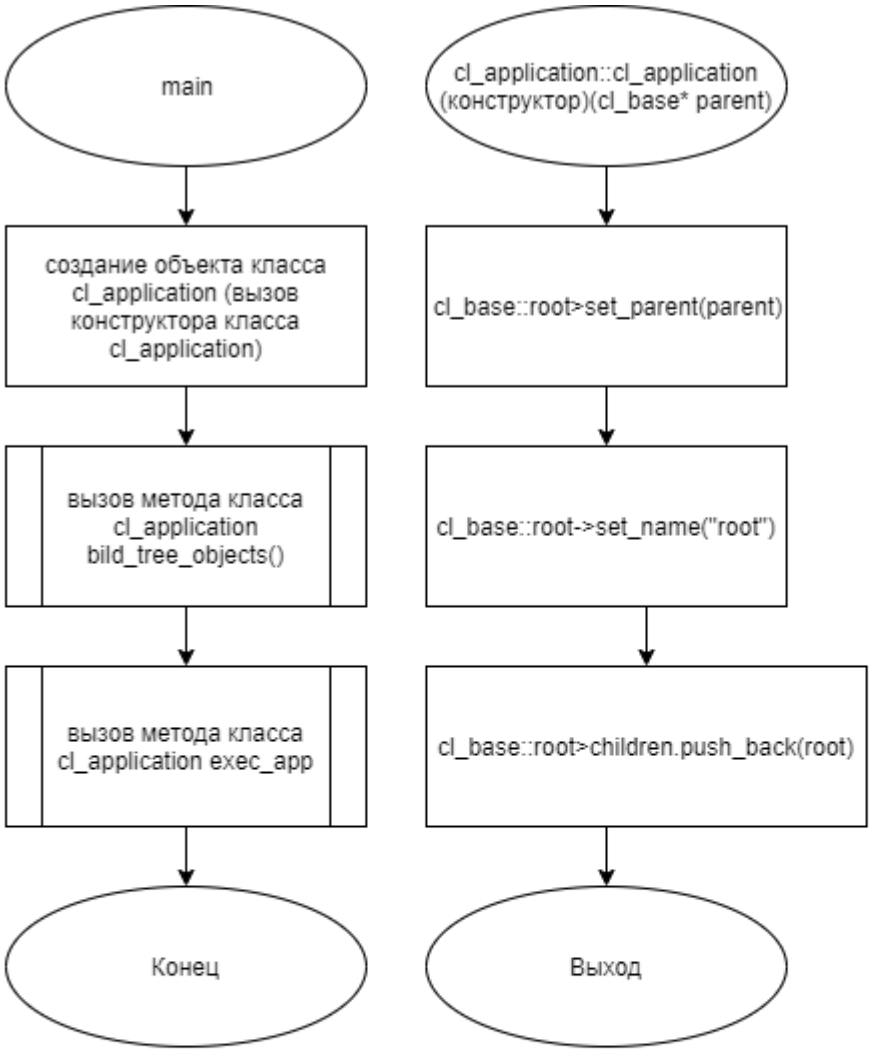
Функционал: вызывает конструктр базового класса

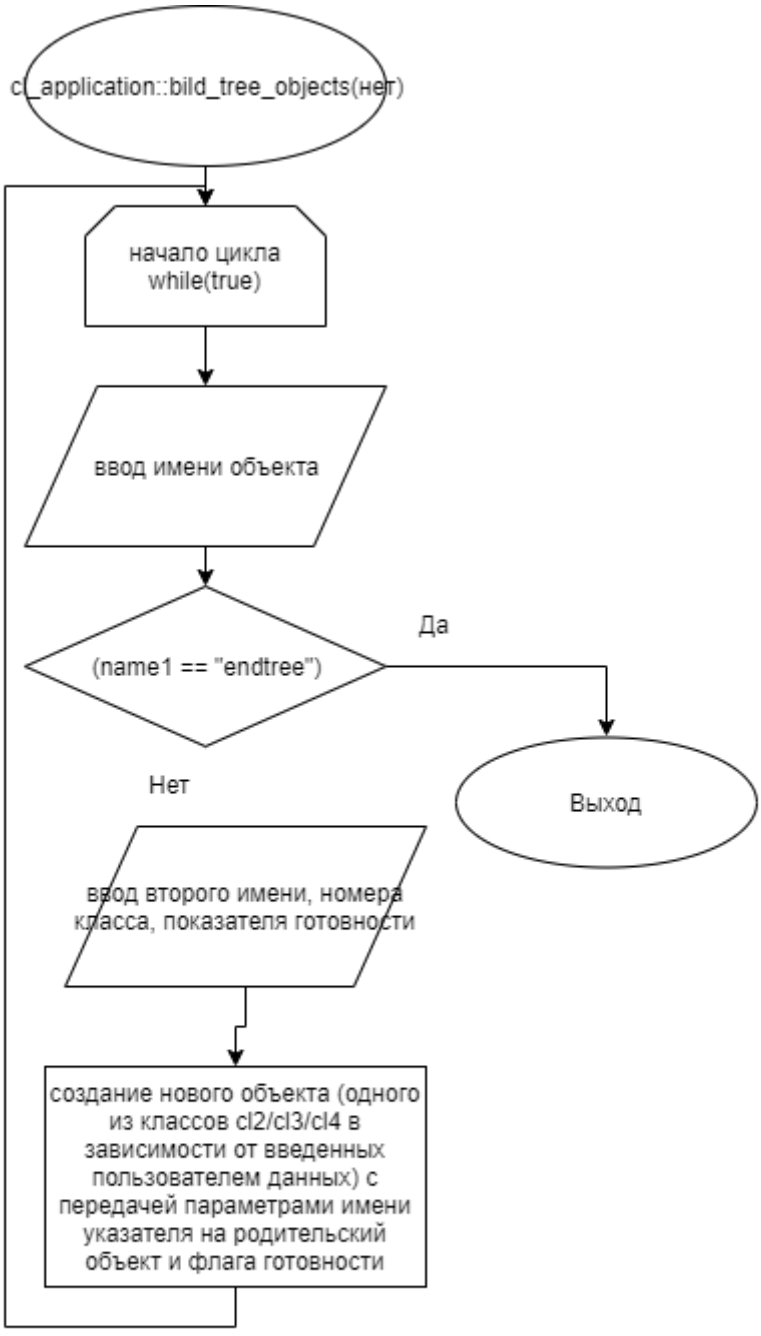
Параметры: string object_name, cl_base* parent, bool ready

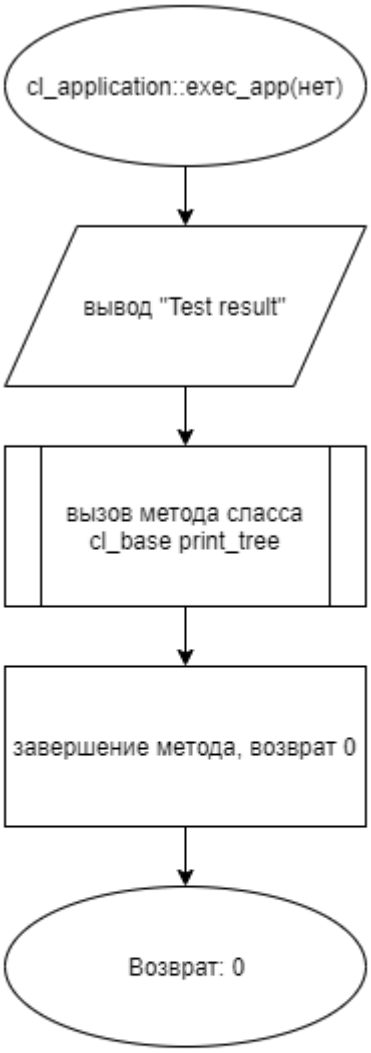
Возвращаемое значение: нет

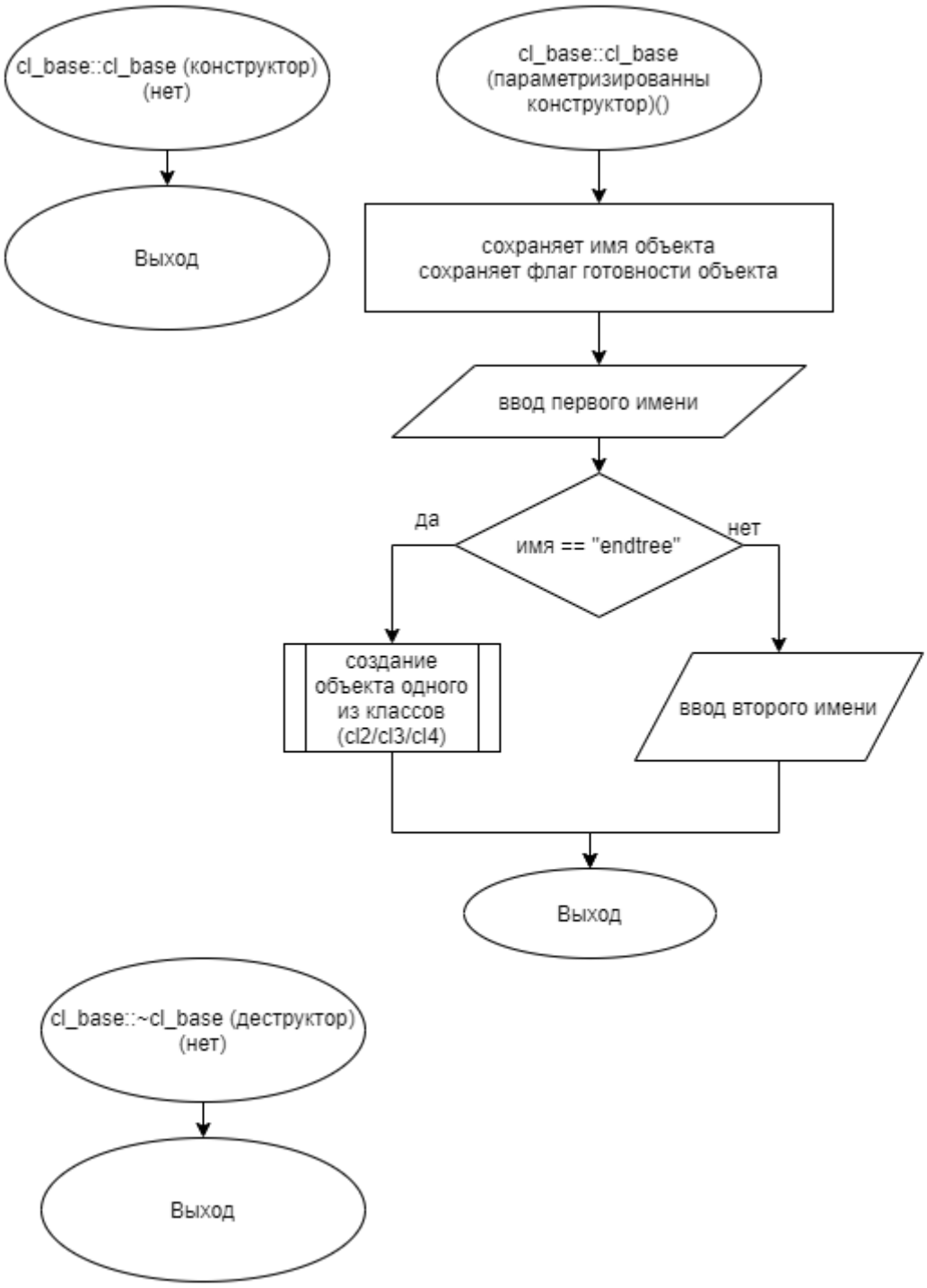
№	Предикат	Действия	№ перехода	Комментарий
1		вызывает конструктр базового класса	∅	

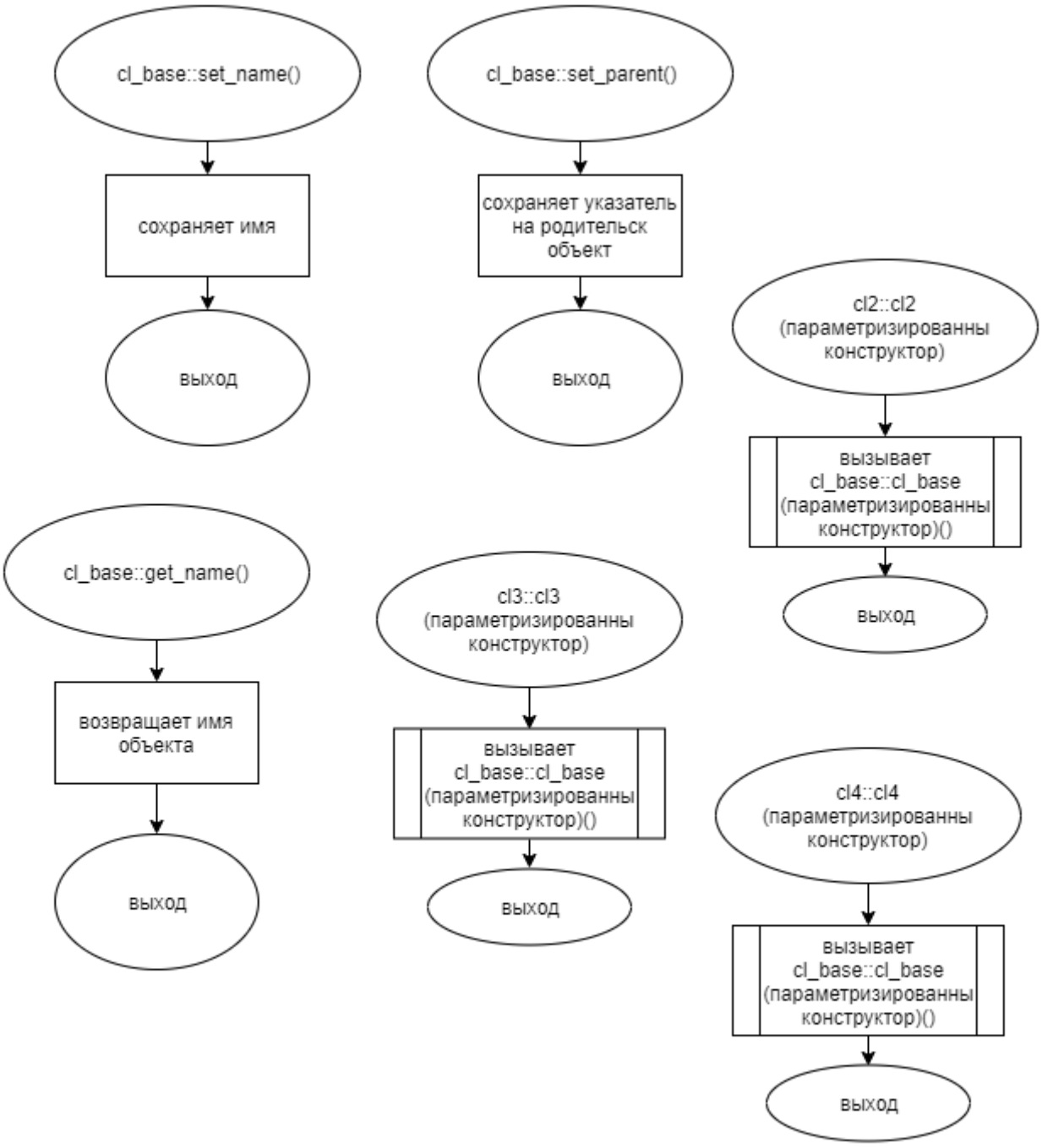
Блок-схема алгоритма

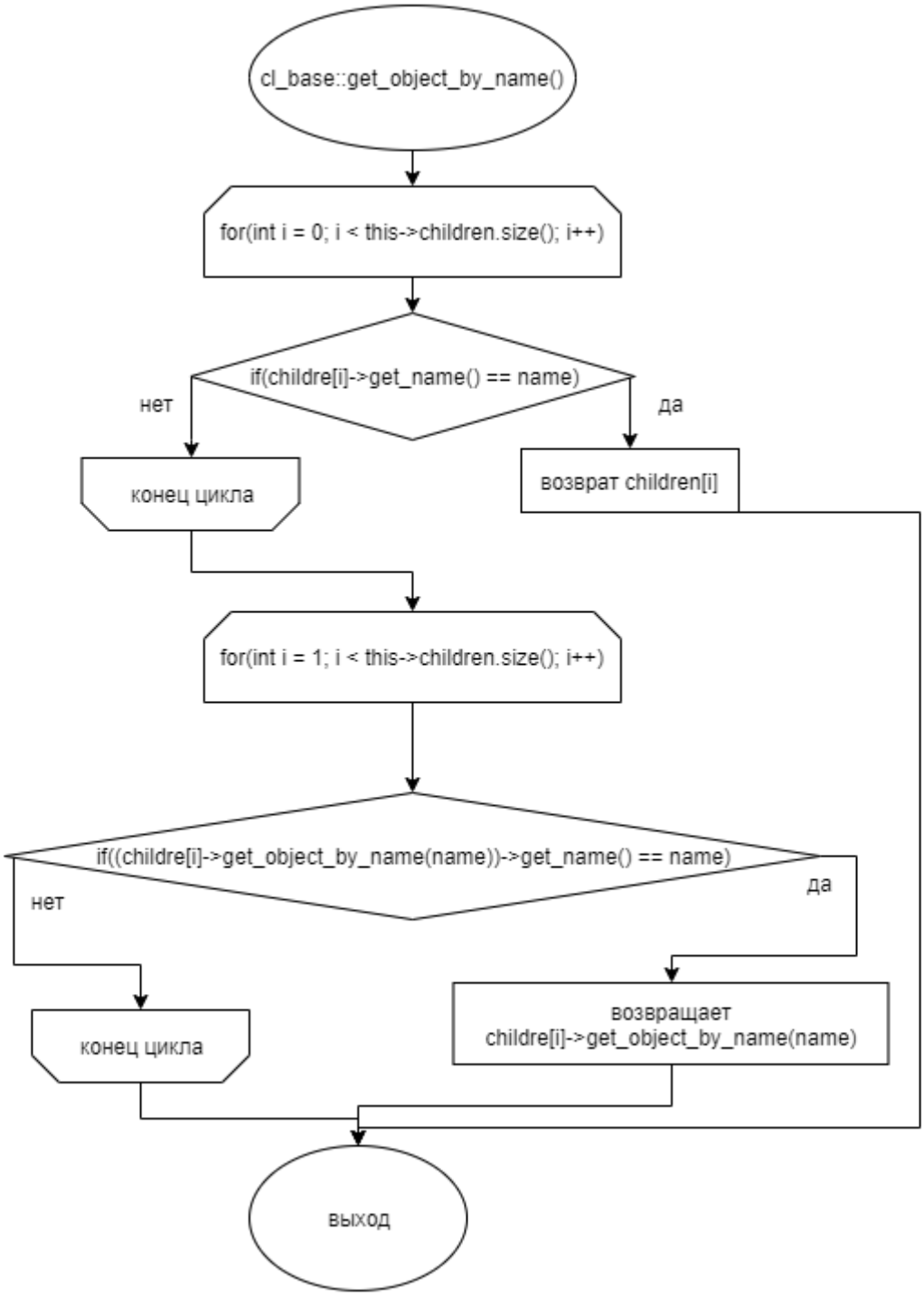


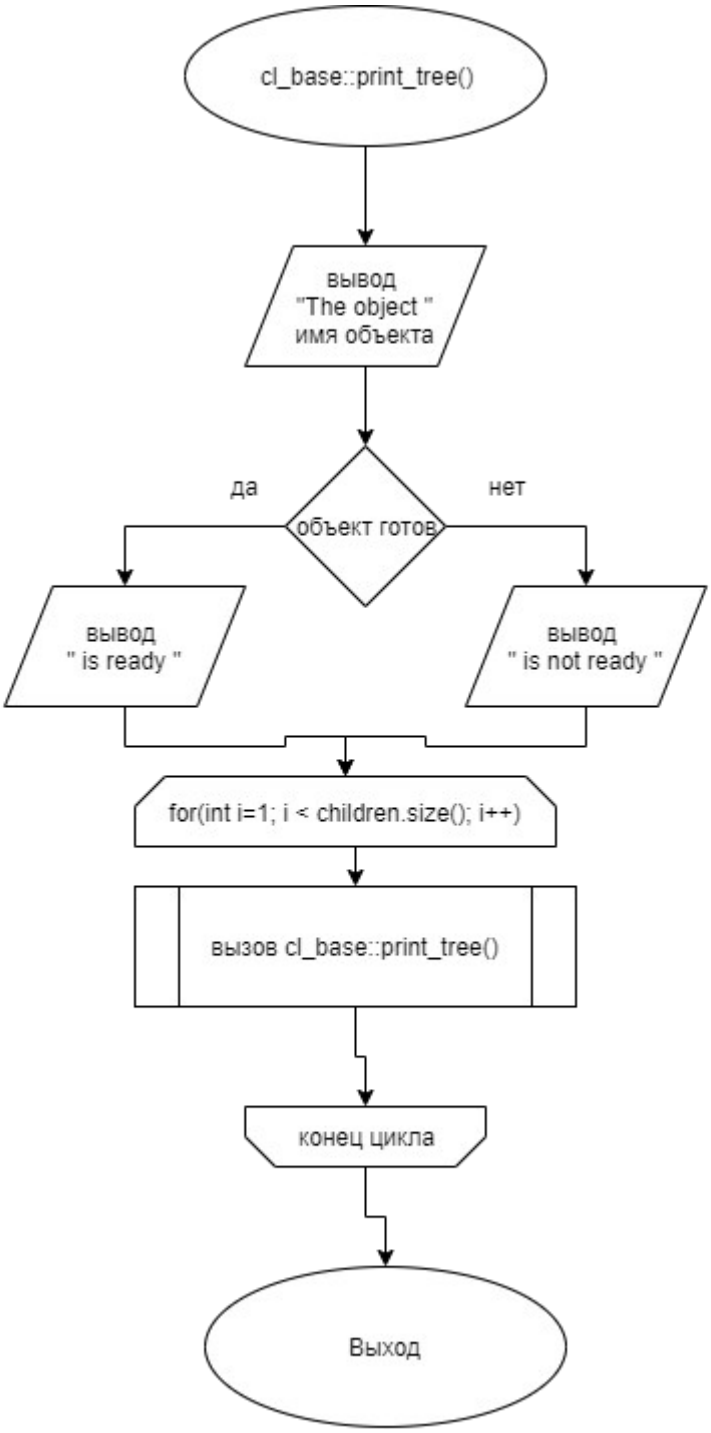

cl_application::bild_tree_objects(нет)
начало цикла while(true)
ввод имени объекта
(name1 == "endtree")
Да
Выход
Нет
ввод второго имени, номера класса, показателя готовности
создание нового объекта (одного из классов cl2/cl3/cl4 в зависимости от введенных пользователем данных) с передачей параметрами имени указателя на родительский объект и флага готовности


cl_application::exec_app(нет)
вывод "Test result"
вызов метода сласса cl_base print_tree
завершение метода, возврат 0
Возврат: 0


cl_base::cl_base (конструктор)
(нет)
Выход
cl_base::cl_base
(параметризированны
конструктор)()
сохраняет имя объекта
сохраняет флаг готовности объекта
ввод первого имени
имя == "endtree"
да
нет
создание
объекта одного
из классов
(cl2/cl3/cl4)
ввод второго имени
Выход
cl_base::~cl_base (деструктор)
(нет)
Выход


cl_base::set_name()
сохраняет имя
выход
cl_base::set_parent()
сохраняет указатель на родительск объект
выход
cl2::cl2 (параметризированны конструктор)
вызывает cl_base::cl_base (параметризированны конструктор)()
выход
cl_base::get_name()
возвращает имя объекта
выход
cl3::cl3 (параметризированны конструктор)
вызывает cl_base::cl_base (параметризированны конструктор)()
выход
cl4::cl4 (параметризированны конструктор)
вызывает cl_base::cl_base (параметризированны конструктор)()
выход


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



Код программы

Файл cl2.h

```
#ifndef CL2_H
#define CL2_H
#include "cl_base.h"
#include "cl_application.h"
class cl2 : public cl_base
{
public:
        cl2(){}
        cl2(string object_name, cl_base* parent, bool ready) :
cl_base(object_name, parent, ready){}
};
#endif
```

Файл cl3.h

```
#ifndef CL3_H
#define CL3_H
#include "cl_base.h"
#include "cl_application.h"
class cl3 : public cl_base
{
public:
        cl3(){}
        cl3(string object_name, cl_base* parent, bool ready) :
cl_base(object_name, parent, ready){}
};
#endif
```

Файл cl4.h

```
#ifndef CL4_H
#define CL4_H
#include "cl_base.h"
#include "cl_application.h"
class cl4 : public cl_base
{
public:
        cl4(){}
        cl4(string object_name, cl_base* parent, bool ready) :
cl_base(object_name, parent, ready){}
};
```

```
#endif
```

Файл cl_application.cpp

```
#include "cl_application.h"
#include "cl2.h"
#include "cl3.h"
#include "cl4.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()
{
        int xz;
        int iready;
        bool ready;
        string name1, name2;
        cin >> name1;
        cl_base* child = new cl_base(name1, nullptr, true);
        while (true)
        {
                cin >> name1;
                if (name1 == "endtree")
                {
                        return;
                }
                cin >> name2 >> xz >> iready;

                if(iready > 0) { ready = true; }
                else { ready = false; }
                cl_base* child2;
                switch(xz)
                {
                        case 2:
                                child2 = new cl2(name2, root-
>get_object_by_name(name1), ready);
                                break;
                        case 3:
                                child2 = new cl3(name2, root-
>get_object_by_name(name1), ready);
                                break;
                        case 4:
                                child2 = new cl4(name2, root-
>get_object_by_name(name1), ready);
                                break;
                }
                //cl_base* child2 = new cl_base(name2, root-
>get_object_by_name(name1), ready);
                //child = child2;
        }
}
```

```
int cl_application::exec_app()
{
        cout << "Test result"<<endl;
        //cout << "The object " << root->children[1]->get_name() << " is
ready";
        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, bool ready)
{
        this->ready = ready;
        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;
}
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));
                }
        }
        return val;
}
void cl_base::print_tree()
{
        cout << "The object " << this->get_name();
        if(this->ready) { cout << " is ready"; }
        else { cout << " is not ready"; }
        for (int i = 1; i < children.size(); i++)
        {
                cout << endl;
                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:
        bool ready;
        int iterator;
        static cl_base* root;
        vector <cl_base*> children;
        cl_base();
        cl_base(string object_name, cl_base* parent, bool ready);
        void set_name(string name);
        void set_parent(cl_base* 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
app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 object_1 object_7 2 1 endtree	Test result The object app_root is ready The object object_1 is ready The object object_7 is ready The object object_2 is ready The object object_4 is not ready The object object_5 is ready The object object_6 is ready The object object_3 is ready	Test result The object app_root is ready The object object_1 is ready The object object_7 is ready The object object_2 is ready The object object_4 is not ready The object object_5 is ready The object object_6 is ready The object object_3 is ready