Постановка задачи

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных программах динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

Построить модель иерархической системы. Реализовать вывод на консоль иерархического дерева объектов в следующем виде:

```
root



    ob_1

    ob_2

    ob_3

    ob_4

    ob
_5



ob
_6



ob
_7
```

где: root - наименование корневого объекта (приложения).

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в контрольной работе № 1.

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в фрагменте методического указания в контрольной работе № 1.

Описание выходных данных

Вывести иерархию объектов в следующем виде:
Object tree
«Наименование корневого объекта»
  «Наименование объекта 1»
    «Наименование объекта 2»
  «Наименование объекта 3»
. . . . .

Отступ каждого уровня иерархии 4 позиции.

Метод решения

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

классы: cl_base, cl_application. cl2, cl3, cl4, cl5

№	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	
		cl5	public		6	
2	cl_application			класс приложения (необходим для запуска и работы самой программы)		
3	cl2			необходим по условию задания		
4	cl3			необходим по условию задания		
5	cl4			необходим по условию задания		
6	cl5			необходим по условию задания		

класс cl_base содержит поля:

1. string object_name - имя объекта
2. cl_base* parent - указатель на объект-родитель
3. bool ready - флаг готовности объекта
4. vector <cl_base*> children - вектор, хранящий в себе указатель на сам объект и его прямых наследников
5. int deep - параметр "глубины" объекта

а так же методы:

1. cl_base(string object_name, cl_base* parent, bool ready) - 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 имени

6. void print_tree() - метод вывода в консоль объектов
7. ~cl_base() - деструктор

класс cl_application содержит методы:

1. cl_application(cl_base* parent) - конструктор
2. void bild_tree_objects() - метод построения дерева объектов
3. int exec_app() - метод итогового вывода

классы: cl2, cl3, cl4, cl5 имеют только конструкторы и необходимы только по условиям задания

Описание алгоритма

Функция: main

Функционал: точка входа

Параметры: нет

Возвращаемое значение: 0

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

Класс объекта: cl_application

Модификатор доступа: public

Метод: cl_application

Функционал: конструтктор

Параметры: cl_base* parent

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		создает объект root (не дступный для пользователя)	∅	

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 быввает)	∅	
2	(name1 == "endtree")	выход	∅	
		ввод второго имени, номера класса, показателя готовности	3	
3		создание объекта (одного из классов cl2/cl3/cl4/cl5 в зависимости от введенных данных) с передачей параметрами имени, указателя на родительский объект и флага готовности	1	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app

Функционал: вывоодт дерево объектов

Параметры: нет

Возвращаемое значение: 0

№	Предикат	Действия	№ перехода	Комментарий
1		вывод "Object tree"	2	
2		вызов метода print_tree	3	
3		завершение метода/возврат 0	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base

Функционал: конструктор

Параметры: string object_name, cl_base* parent, bool ready

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет полученные параметры в поля объекта	2	
2	(parent != nullptr)	сохраняет указатель на себя в vector объекта-родителя	∅	
		нечего	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_patrent

Функционал: сохраняет указаетль на объект родиетль

Параметры: cl_base* parent

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет указаетль на объект родиетль	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_name

Функционал: сохраняет имя объекта

Параметры: string name

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет имя объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_name

Функционал: возвращает имя объекта

Параметры: нет

Возвращаемое значение: string name

№	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)	возврат children[i] выход	∅	
		переход	1	
3	(i<chuldren.size())	вызов get_object_by_name от именидочернего объекта	4	
		выход	∅	
4	((children[i]->getobject_by_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<deep	вывод "\t"	1	
		переход	2	
2		вывод имени объекта	3	
3	j<children.size	вызывает print_tree от имени наследника j	3	
		выход	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: ~cl_base

Функционал: деструктор

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Класс объекта: cl2

Модификатор доступа: public

Метод: cl2

Функционал: вызывает конструктро базового класса

Параметры: string object_name, cl_base* parent, bool ready

Возвращаемое значение: нет

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

Класс объекта: cl3

Модификатор доступа: public

Метод: cl3

Функционал: вызывает конструктро базового класса

Параметры: string object_name, cl_base* parent, bool ready

Возвращаемое значение: нет

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

Класс объекта: cl4

Модификатор доступа: public

Метод: cl4

Функционал: вызывает конструктро базового класса

Параметры: string object_name, cl_base* parent, bool ready

Возвращаемое значение: нет

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

Класс объекта: cl5

Модификатор доступа: public

Метод: cl5

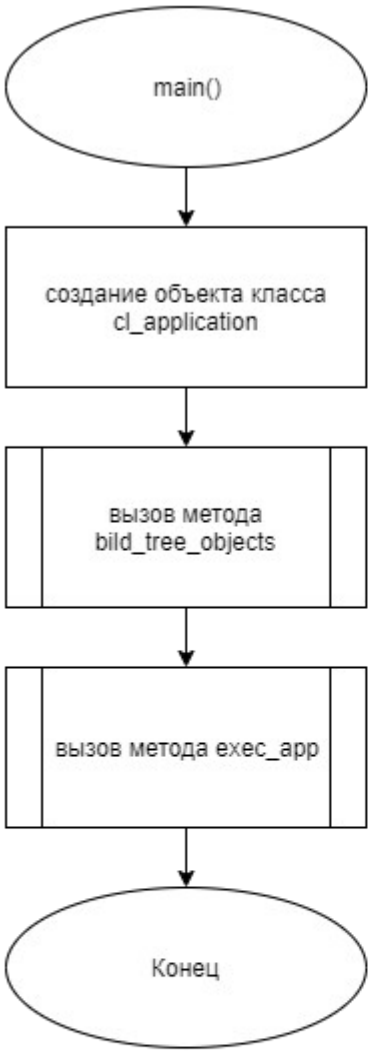
Функционал: вызывает конструктро базового класса

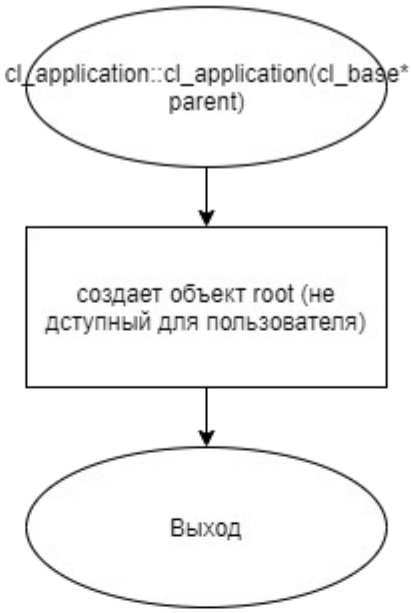
Параметры: string object_name, cl_base* parent, bool ready

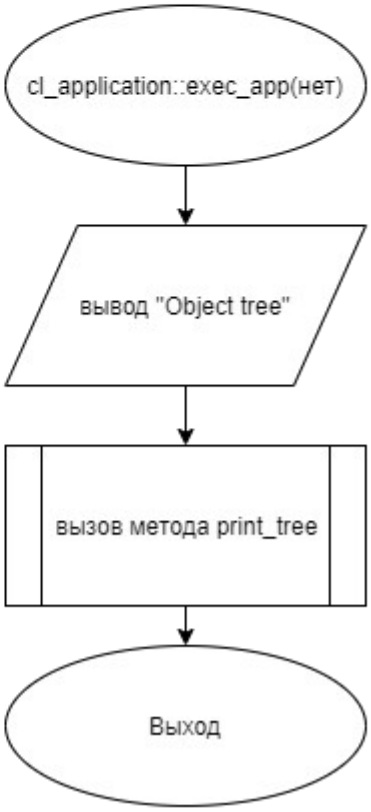
Возвращаемое значение: нет

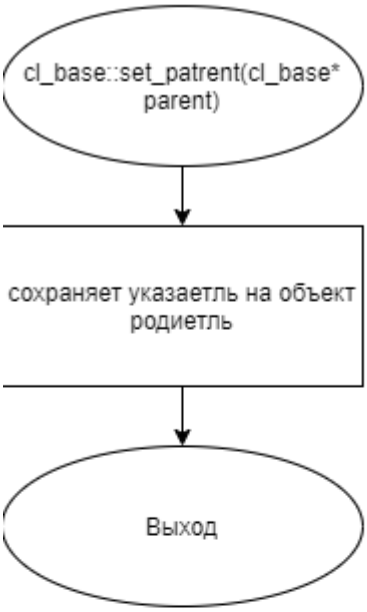
№	Предикат	Действия	№ перехода	Комментарий
1		вызывает конструктро базового класса	∅	

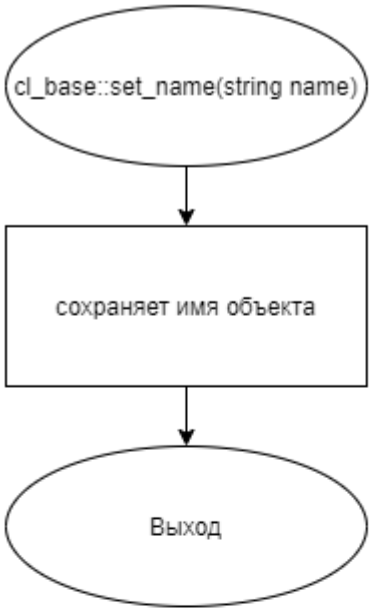
Блок-схема алгоритма

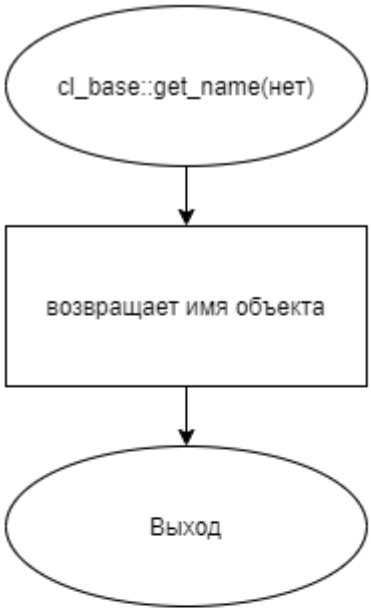

main()
создание объекта класса cl_application
вызов метода bild_tree_objects
вызов метода exec_app
Конец

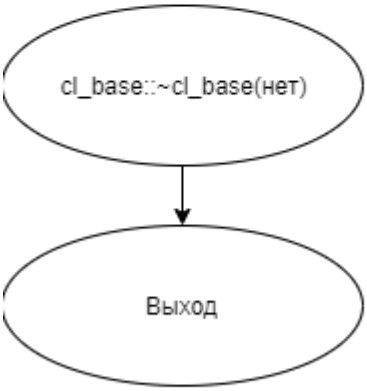

cl_application::cl_application(cl_base* parent)
создает объект root (не дступный для пользователя)
Выход

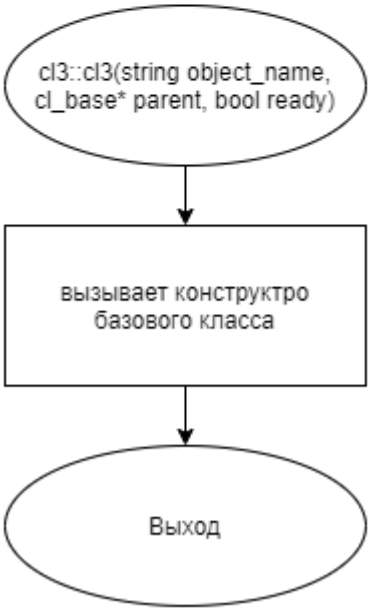

cl_application::exec_app(нет)
вывод "Object tree"
вызов метода print_tree
Выход

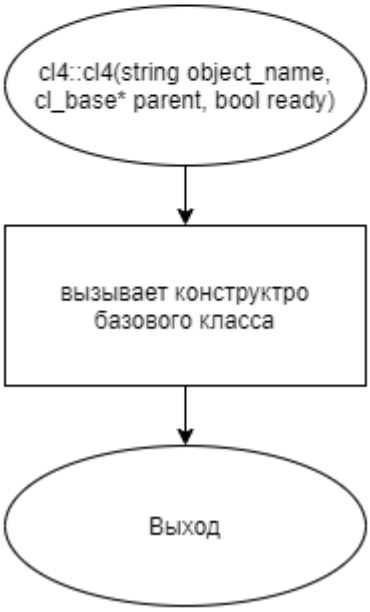

cl_base::set_patrent(cl_base* parent)
сохраняет указаетль на объект родиетль
Выход

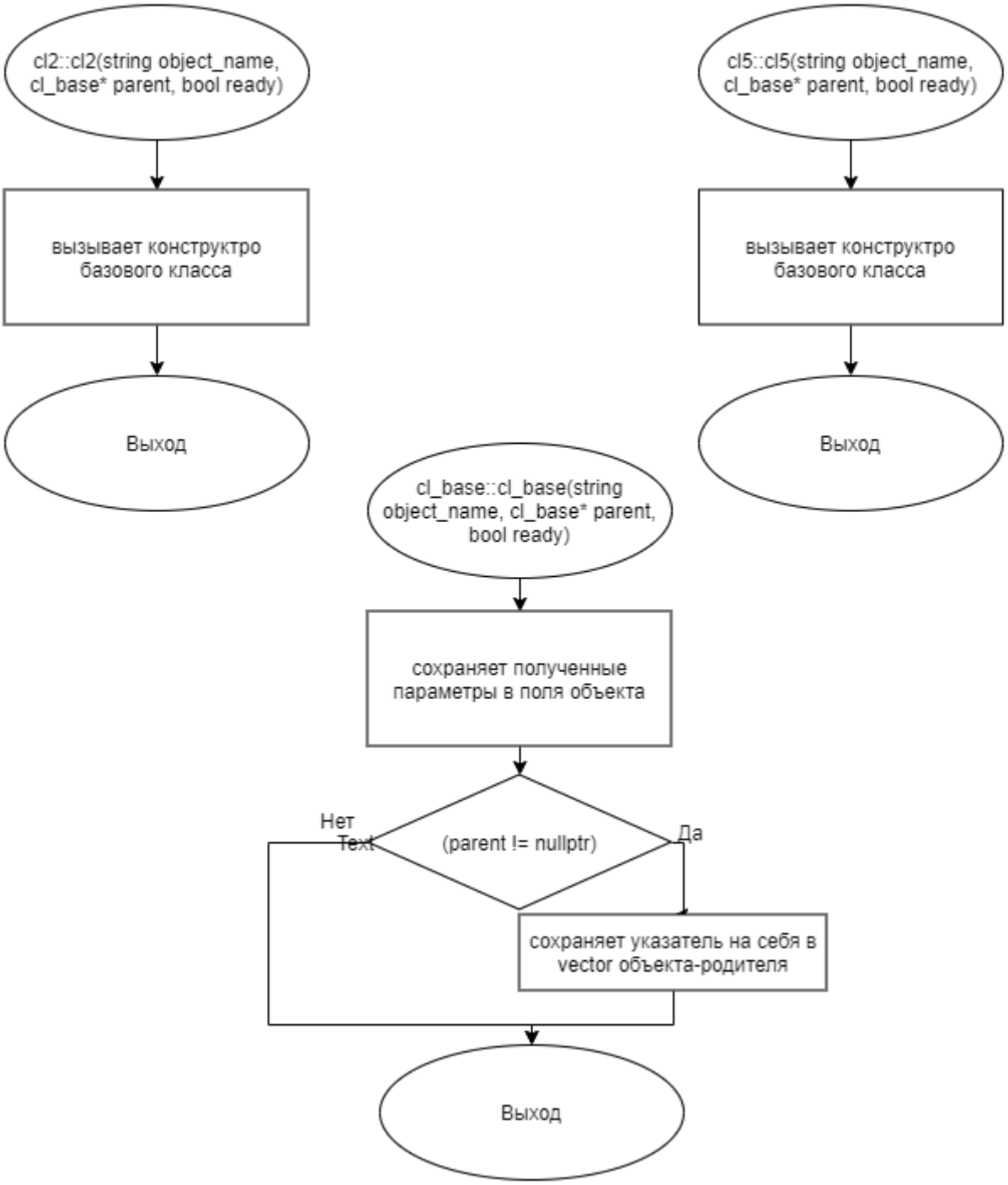

cl_base::set_name(string name)
сохраняет имя объекта
Выход

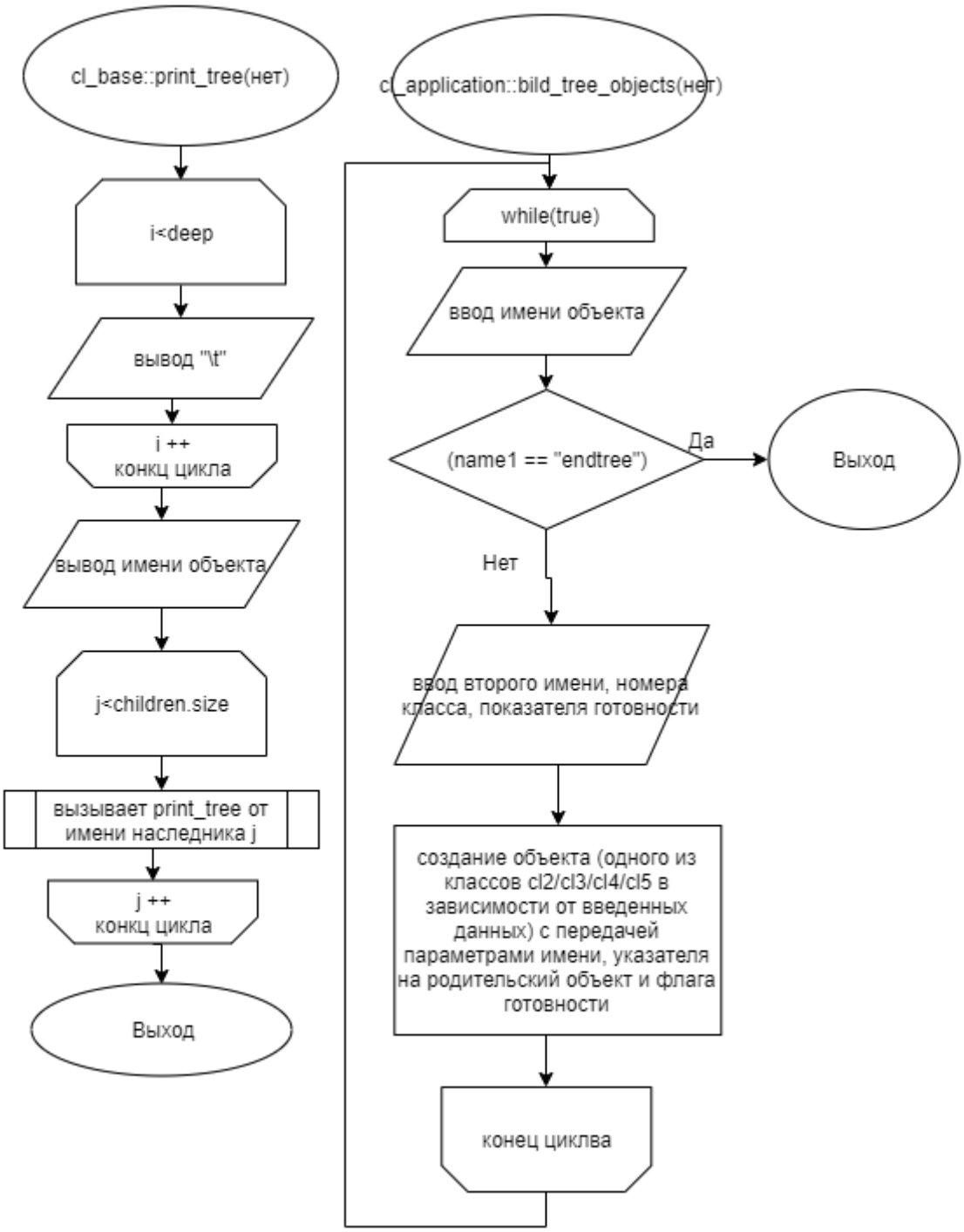

cl_base::get_name(нет)
возвращает имя объекта
Выход

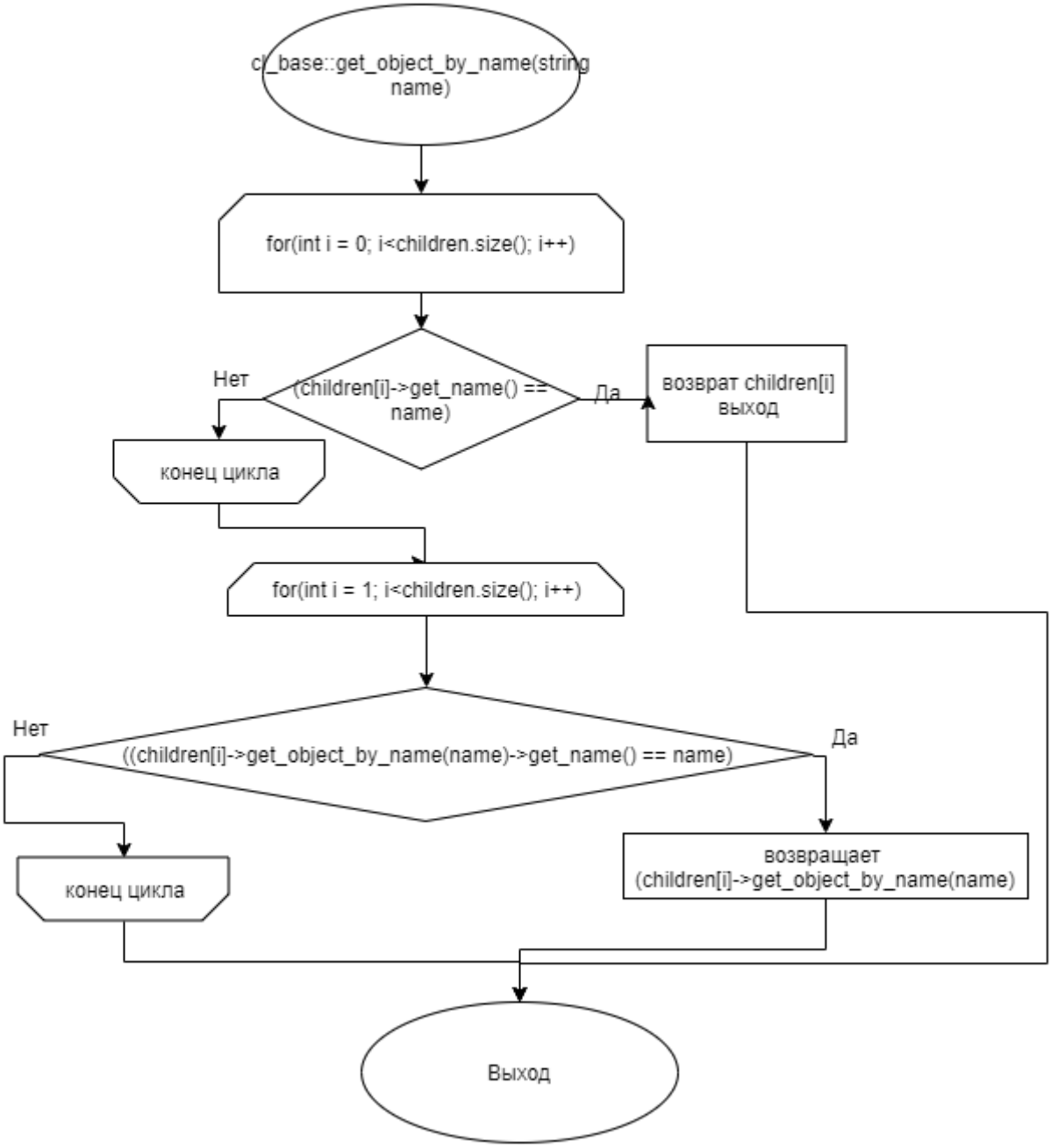

cl_base::~cl_base(нет)
Выход


cl3::cl3(string object_name, cl_base* parent, bool ready)
вызывает конструктро базового класса
Выход


cl4::cl4(string object_name, cl_base* parent, bool ready)
вызывает конструктро базового класса
Выход


cl2::cl2(string object_name, cl_base* parent, bool ready)
вызывает конструктро базового класса
Выход
cl5::cl5(string object_name, cl_base* parent, bool ready)
вызывает конструктро базового класса
Выход
cl_base::cl_base(string object_name, cl_base* parent, bool ready)
сохраняет полученные параметры в поля объекта
(parent != nullptr)
Нет
Text
Да
сохраняет указатель на себя в vector объекта-родителя
Выход


cl_base::print_tree(нет)
i<deep
вывод "\t"
i ++
конкц цикла
вывод имени объекта
j<children.size
вызывает print_tree от имени наследника j
j ++
конкц цикла
Выход
cl_application::bild_tree_objects(нет)
while(true)
ввод имени объекта
(name1 == "endtree")
Да
Выход
Нет
ввод второго имени, номера класса, показателя готовности
создание объекта (одного из классов cl2/cl3/cl4/cl5 в зависимости от введенных данных) с передачей параметрами имени, указателя на родительский объект и флага готовности
конец циклва


cl_base::get_object_by_name(string name)
for(int i = 0; i<children.size(); i++)
(children[i]->get_name() == name)
Нет
Да
возврат children[i] выход
конец цикла
for(int i = 1; i<children.size(); i++)
((children[i]->get_object_by_name(name)->get_name() == name)
Нет
Да
конец цикла
возвращает (children[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 cl5.h

```
#ifndef CL5_H
#define CL5_H
#include "cl_base.h"
#include "cl_application.h"
class cl5 : public cl_base
{
public:
        cl5(){}
        cl5(string object_name, cl_base* parent, bool ready) :
cl_base(object_name, parent, ready){}
};
#endif
```

Файл cl6.h

```
#ifndef CL6_H
#define CL6_H
#include "cl_base.h"
#include "cl_application.h"
class cl6 : public cl_base
{
public:
        cl6(){}
        cl6(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"
#include "cl5.h"
#include "cl6.h"
cl_application::cl_application(cl_base* parent = nullptr)
{
        cl_base::root->set_parent(parent);
        cl_base::root->set_name("r01oot");
        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;
                        case 5:
                                child2 = new cl5(name2, root-
>get_object_by_name(name1), ready);
                                break;
                        case 6:
                                child2 = new cl6(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 << "Object tree"<<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->deep = parent->deep + 1;
                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()
{
        for(int i = 0; i < deep; i++)
        {
                cout << "    ";
        }
        cout << this->get_name();
        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;
        int deep = 0;
        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();
}
/*
root
root ob_1 3 1
root ob_3 4 2
ob_1 ob_2 2 -1
ob_3 ob_6 2 1
ob_3 ob_4 6 1
ob_4 ob_5 5 1
ob_6 ob_7 2 -7
endtree
*/
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
roo roo ob1 2 1 roo ob2 3 1 ob1 ob5 4 1 endtree	Object tree roo ob1 ob5 ob2	Object tree roo ob1 ob5 ob2