МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« КЛ_3_2 Вывод иерархического дерева »

С тудент группы	ИНБО-15-20	Ло В.Х.
Руководитель практики	Ассистент	Рогонова О.Н.
Работа представлена	«___»_____2020 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2020

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


                                                                  11
    ob_1
                                                                  15
    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

Потоки ввод/вывод cin/cout.

Объект класса: cl_base

1.Описание класса: cl_base

Свойства:

+ наименование объекта: строкового типа

+ список указателей на объекты плеченных к текущему объекту в дереве иерахии

Функционал:

-параметриванный конструкор с параметрами

-определения имени объекта

-получения имени объекта

-получения указателя на головной объект текущего объекта

-токлнуть объектов в конец векторов

-выбирять объеков в дереве иерахии

-установить состояния объекта

-получить состояния объектов

2.Описание класса: cl_application

Наименование класса объекта: cl_application

Функционал:

-построение дерева объекта

-запуск системы

№	Имя класса	класса следник	Модификатор доступа при наследовании	Описание	номер
1				Базовый класса виерархииклассов. Содержит основный поля и методы	
	cl_base	cl_application	public		2
		cl_1	public		
		cl_2	public		
		cl_3	public		
		cl_4	public		
		cl_5	public		
2	cl_applicati			класс корневого объекта	

	on				

Описание алгоритма

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base()

Функционал: параметризиванные парамеиры

Параметры: cl_base* parent,string name

Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий
1		параметризиванные парамеиры	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_object_name()

Функционал: определения имени объекта

Параметры: string name

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		определения имени объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_object_name()

Функционал: получения имени объекта

Параметры: нет

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		получения имени объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_parent()

Функционал: переопределение объекта для текущего в дереве иерахии

Параметры: cl_base* parent

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		переопределение объекта для текущего в дереве иерахии	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_parent()

Функционал: получения указателя на головой объект текущего обекта

Параметры: нет

Возвращаемое значение: parent

№	Предикат	Действия	№ перехода	Комментарий
1		получения указателя на головой объект текущего обекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_status()

Функционал: вводить номер исходного состояния очередного объекта

Параметры: status

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		вводить номер исходного состояния очередного объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_by_name()

Функционал: находим элемент с именем

Параметры: string name

Возвращаемое значение: nullptr

№	Предикат	Действия	№ перехода	Комментарий
1		находим элемент с именем	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: show_tree_view()

Функционал: вводит информацию о вершинах в виде дерева

Параметры: deep_lvl

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		int indent= deep_lvl* 4;	2	
2	i<indent	cout<<" "	3	
			4	
3		++i	2	
4		cout<<this->name	5	
5	!this->children.size()		∅	
			6	
6		cout<<endl	7	
7	i<this->children.size()	this->children[i]->show_tree_view(deep_lvl +1);	8	
			∅	
8	i!= this-		9	

	>children.size()-1	cout<<endl		
			9	
9		++i	7	

Класс объекта: cl_application

Модификатор доступа: public

Метод: bild_tree_objects()

Функционал: построение дерева объекта

Параметры: нет

Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий
1		cl_1* ob_1; cl_2* ob_2; cl_3* ob_3; cl_4* ob_4; cl_5* ob_5; string nameParent,nameChild; int class_type = 0,status=1;	2	
2		ввод nameParent	3	
3		this->set_object_name(nameParent)	4	
4	true	ввод>>nameParent	5	
			∅	
5	nameParent=="endtree"	break	∅	
			6	
6		cin>>nameChild>>class_type>>status;	7	
7		cl_base* ob_3=this->get_by_name(nameParent);	8	
8	class_type==2	ob_1=new cl_1(get_by_name(nameParent),nameChild); ob_1->set_status(status);	5	
			9	
9	class_type==3	ob_2=new cl_2(get_by_name(nameParent),nameChild); ob_2->set_status(status);	5	
			10	
10	class_type==4	ob_3=new	5	

		cl_3(get_by_name(nameParent),nameChild); ob_3->set_status(status);		
			11	
11	class_type==5	ob_4=new cl_4(get_by_name(nameParent),nameChild); ob_4->set_status(status);	5	
			12	
12	class_type==6	ob_5=new cl_5(get_by_name(nameParent),nameChild); ob_5->set_status(status);	5	
			5	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app()

Функционал: вынести соответствующую информацию

Параметры: нет

Возвращаемое значение: int код, возврата

№	Предикат	Действия	№ перехода	Комментарий
1		cout<<"Object tree"<<endl	2	
2		this->show_tree_view(0)	∅	

Функция: main

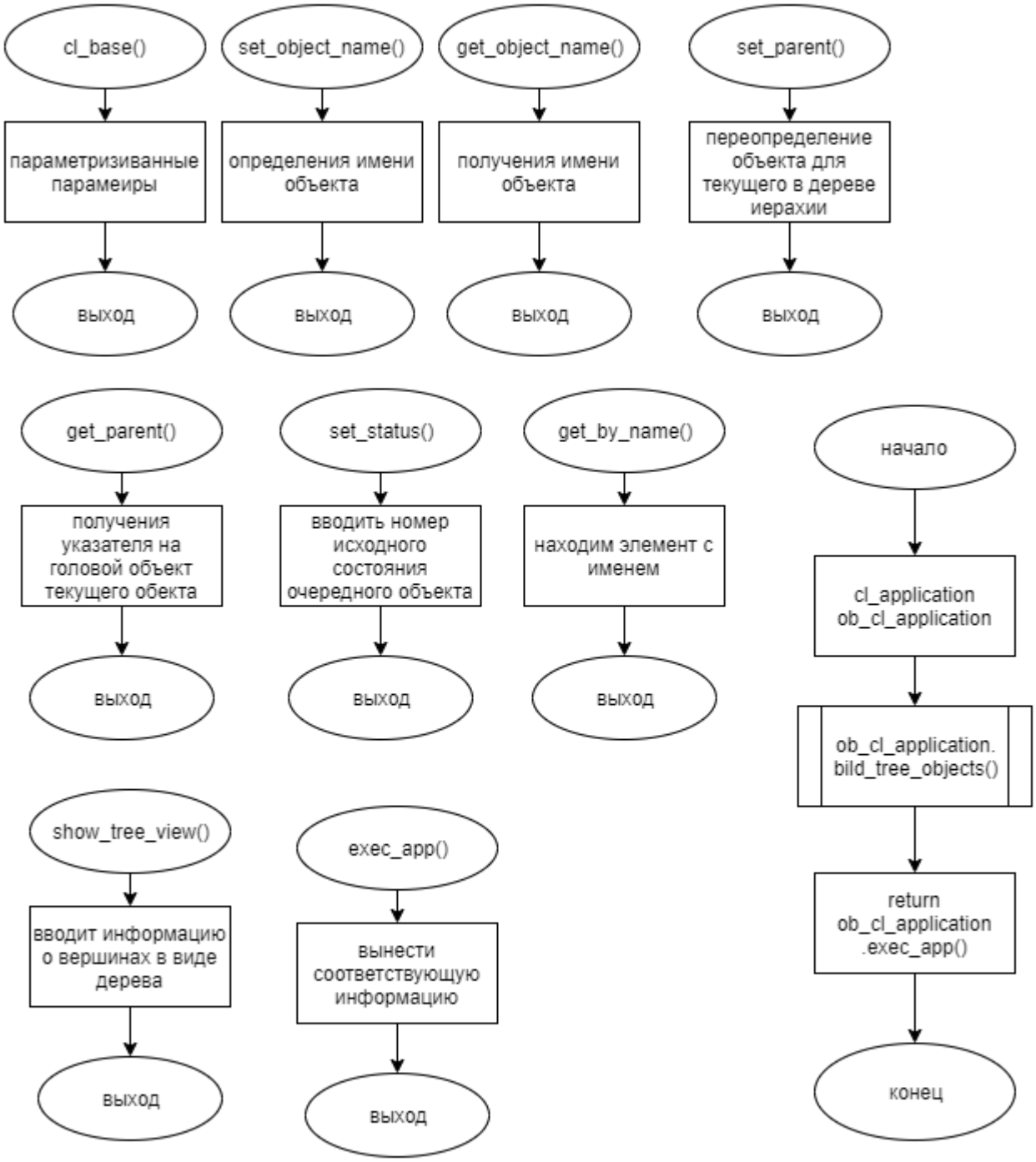
Функционал: основнная программа

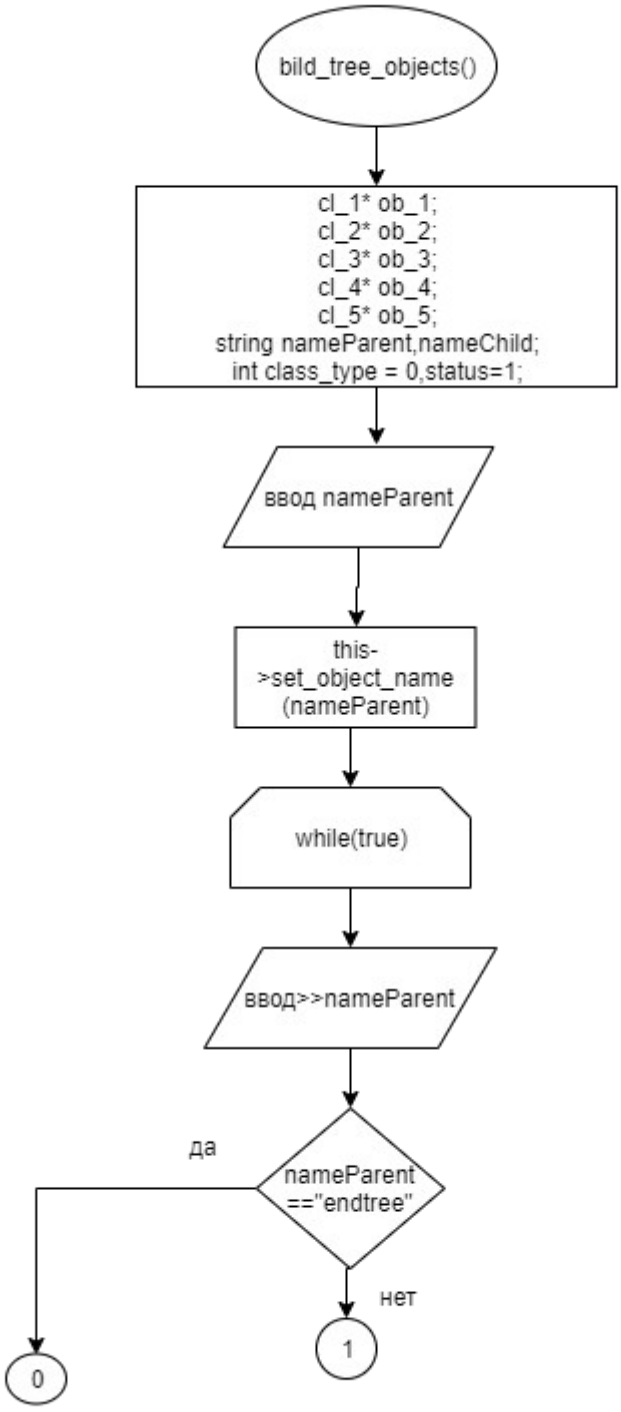
Параметры: нет

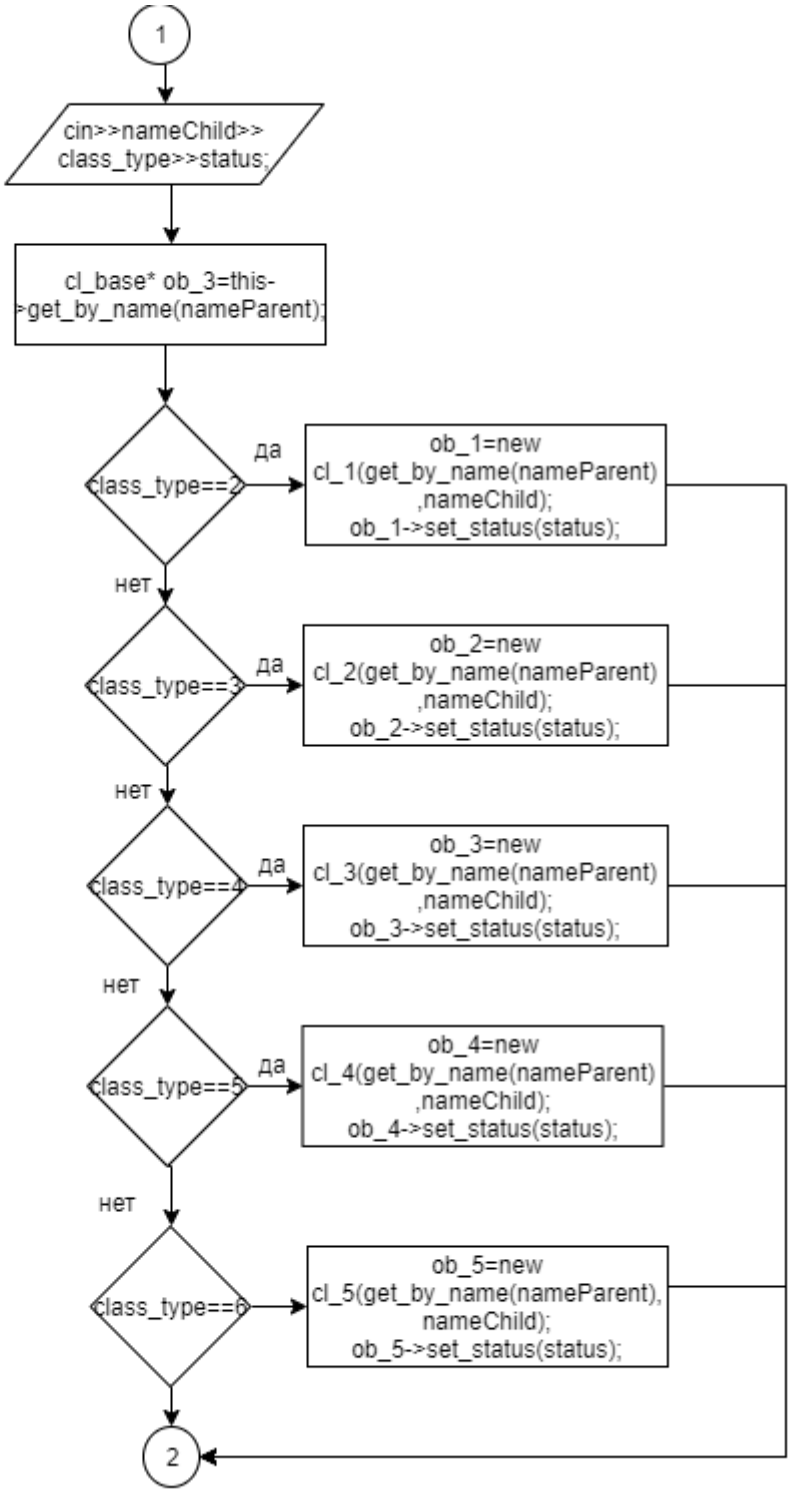
Возвращаемое значение: int код, возврата

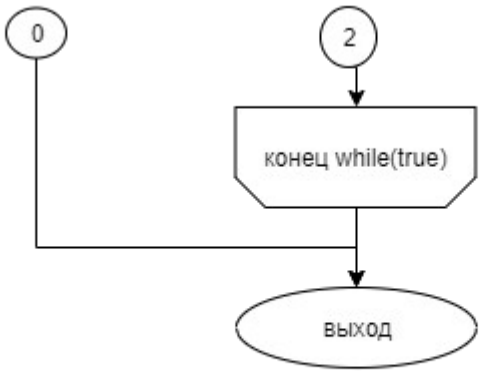
№	Предикат	Действия	№ перехода	Комментарий
1		cl_application ob_cl_application	2	
2		ob_cl_application.bild_tree_objects()	3	
3		return ob_cl_application.exec_app()	∅	

Блок-схема алгоритма




bild_tree_objects()
cl_1* ob_1;
cl_2* ob_2;
cl_3* ob_3;
cl_4* ob_4;
cl_5* ob_5;
string nameParent,nameChild;
int class_type = 0,status=1;
ввод nameParent
this-
>set_object_name
(nameParent)
while(true)
ввод>>nameParent
да
nameParent
=="endtree"
нет
1
0


1
cin>>nameChild>>
class_type>>status;
cl_base* ob_3=this-
>get_by_name(nameParent);
class_type==2
да
ob_1=new
cl_1(get_by_name(nameParent)
,nameChild);
ob_1->set_status(status);
нет
class_type==3
да
ob_2=new
cl_2(get_by_name(nameParent)
,nameChild);
ob_2->set_status(status);
нет
class_type==4
да
ob_3=new
cl_3(get_by_name(nameParent)
,nameChild);
ob_3->set_status(status);
нет
class_type==5
да
ob_4=new
cl_4(get_by_name(nameParent)
,nameChild);
ob_4->set_status(status);
нет
class_type==6
ob_5=new
cl_5(get_by_name(nameParent),
nameChild);
ob_5->set_status(status);
2


0
2
конец while(true)
выход

Код программы

Файл cl_1.h

```
#ifndef CL_1_H
#define CL_1_H
#include<string>
#include "cl_base.h"
class cl_1 : public cl_base
{
public:
cl_1(cl_base* parent,std::string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_2.h

```
#ifndef CL_2_H
#define CL_2_H
#include<string>
#include "cl_base.h"
class cl_2 : public cl_base
{
public:
cl_2(cl_base* parent,std::string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_3.h

```
#ifndef CL_3_H
#define CL_3_H
#include<string>
#include "cl_base.h"
class cl_3 : public cl_base
{
public:
cl_3(cl_base* parent,std::string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_4.h

```
#ifndef CL_4_H
#define CL_4_H
#include<string>
#include "cl_base.h"
class cl_4 : public cl_base
{
public:
cl_4(cl_base* parent,std::string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_5.h

```
#ifndef CL_5_H
#define CL_5_H
#include<string>
#include "cl_base.h"
class cl_5 : public cl_base
{
public:
cl_5(cl_base* parent,std::string name) : cl_base(parent,name){}
};
#endif
```

Файл cl_application.cpp

```
#include<iostream>
#include "cl_application.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_base.h"
using namespace std;
void cl_application::bild_tree_objects()
{
        cl_1* ob_1;
```

```
        cl_2* ob_2;
        cl_3* ob_3;
        cl_4* ob_4;
        cl_5* ob_5;
        string nameParent,nameChild;
        int class_type = 0,status=1;
        cin>>nameParent;
        this->set_object_name(nameParent);
        while(true){
                cin>>nameParent;
                if(nameParent=="endtree"){
                        break;
                }
                cin>>nameChild>>class_type>>status;
                cl_base* ob_3=this->get_by_name(nameParent);
                if(class_type==2)
                {
                        ob_1=new cl_1(get_by_name(nameParent),nameChild);
                        ob_1->set_status(status);
                }
                else if(class_type==3)
                {
                        ob_2=new cl_2(get_by_name(nameParent),nameChild);
                        ob_2->set_status(status);
                }
                else if(class_type==4)
                {
                        ob_3=new cl_3(get_by_name(nameParent),nameChild);
                        ob_3->set_status(status);
                }
                else if(class_type==5)
                {
                        ob_4=new cl_4(get_by_name(nameParent),nameChild);
                        ob_4->set_status(status);
                }
                else if(class_type==6)
                {
                        ob_5=new cl_5(get_by_name(nameParent),nameChild);
                        ob_5->set_status(status);
                }

        }
}
int cl_application::exec_app()
{
        cout<<"Object tree"<<endl;
        this->show_tree_view(0);
        return 0;
}
```

Файл cl_application.h

```
#ifndef CL_APPLICATION_H
```

```
#define CL_APPLICATION_H
#include "cl_base.h"
using namespace std;
class cl_application : public cl_base
{
public:
                void bild_tree_objects();
                int exec_app();
};
#endif
```

Файл cl_base.cpp

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

void cl_base::set_parent(cl_base* parent)
{
        if(parent)
        {
                if(this->parent)
                {
```

```
                        for(int i = 0 ; i<this->parent->children.size();i++)
                        {
                                if(this->parent->children[i] == this)
                                {
                                        this->parent-
>children.erase(children.begin()+i);
                                }
                        }
                }
                this->parent = parent;
                this->parent->children.push_back(this);
        }
}
cl_base* cl_base:: get_parent()
{
        return this->parent;
}
void cl_base::show_tree()
{
        if(!this->children.size())
                return;
                cout<<this->name;
        for(int i=0;i < this->children.size();++i)
        {
                cout<<"  ";
                cout<<this->children[i]->get_object_name();
        }
        bool w=false;
        for(int i=0;i<this->children.size();++i)
        {
                if(this->children[i]->children.size() && !w)
                {
                        cout<<"\n";
                        w=true;
                }
                this->children[i]->show_tree();
        }
        return;
}
void cl_base::show_status_tree()
{
        cout<<"The object "<<this->name;
        if(this->status>0){
                cout<<" is ready";
        }
        else
        {
                cout<<" is not ready";
        }
        if(this->children.size())
        {
                cout<<"\n";
        }
        for(int i=0;i<this->children.size();++i)
        {
                this->children[i]->show_status_tree();
                if(i!=this->children.size()-1)
                        cout<<endl;
        }
        return;
```

```
}
void cl_base::show_tree_view(int deep_lvl=0)
{
        int indent= deep_lvl* 4;
        for(int i=0;i<indent;++i)
        {
                cout<<" ";
        }
        cout<<this->name;
        if(!this->children.size())
                return;
        cout<<endl;
        for(int i=0;i<this->children.size();++i)
        {
                this->children[i]->show_tree_view(deep_lvl +1 );
                if(i!= this->children.size()-1)
                {
                        cout<<endl;
                }
        }
        return;
}
cl_base* cl_base::get_by_name(string name)
{
        if(this->name==name){
                return this;
        }
        for(int i=0;i<this->children.size();++i)
        {
                cl_base* cl_3=children[i]->get_by_name(name);
                if(cl_3!=NULL)
                        return cl_3;
        }
        return nullptr;
}
void cl_base::set_status(int status)
{
this->status=status;
}
```

Файл cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H
#include<iostream>
#include<vector>
#include<string>
using namespace std;
```

```
class cl_base
{
protected:
        vector <cl_base*> children;
        cl_base* parent;
        string name;
        int status;
public:
        cl_base(cl_base* parent= nullptr,string name="cl_base");
        ~cl_base();
        void set_object_name(string name);
        string get_object_name();
        void set_parent(cl_base* parent);
        cl_base* get_parent();
        void show_tree();
        void show_status_tree();
        void show_tree_view(int deep_lvl);
        cl_base* get_by_name(string name);
        void set_status(int status);
};
#endif
```

Файл main.cpp

```
#include "cl_application.h"
int main()
{
cl_application ob_cl_application;
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	Object tree app_root object_1	Object tree app_root object_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 object_7 object_8 2 1 endtree	object_7 object_8 object_2 object_4 object_5 object_6 object_3	object_7 object_8 object_2 object_4 object_5 object_6 object_3