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_1 Проверка готовности объектов к работе »

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

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

Потоки ввод/вывод(cin/cout).

1.Объект класса: cl_base

Описание класса: cl_base

Свойства:

+ наименование объекта: строкового типа

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

Функционал:

-параметриванный конструкор с параметрами

-определения имени объекта

-получения имени объекта

-получения указтеля на головной объект текущего объекта

-токлнуть объект в конец вектора

-выбирять обектов в дереве иерархии

-установить состояния объекта

-получить состояния объектов

2.Описание объекта: 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 в иерархии классов. Содержит основный поля и методы	

	Base	cl_application	public		2
		cl_2	public		
		cl_3	public		
		cl_4	public		
2	cl_application			класс корневого объекта	

Описание алгоритма

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base()

Функционал: конструктор класса

Параметры: cl_base* parent,string name

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1		параметризиванные параметры	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_object_name()

Функционал: опередделения имени объекта

Параметры: string name

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий

1		опеределения 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 иерархии

Параметры: cl_base* parent

Возвращаемое значение: нет

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

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_parent()

Функционал: получения указателя на головной объект текущего обекта

Параметры: нет

Возвращаемое значение: parent

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

Класс объекта: cl_base

Модификатор доступа: public

Метод: show_tree()

Функционал: вывода наименнований объектов в дереве иерахии слева направои сверху вниз

Параметры: Нет

Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий
1		вывода наименнований объектов в дереве иерахии слева направои сверху вниз	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: show_status_tree()

Функционал: Выводит статусы вершин поддерена из данной вершины

Параметры: Нет

Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий
1		Выводит статусы вершин поддерена из данной вершины	∅	0

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_by_name

Функционал: находим елемент с имнем

Параметры: string name

Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий
1		находим елемент с имнем	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app()

Функционал: запуск системы

Параметры: Нет

Возвращаемое значение: код , возврата

№	Предикат	Действия	№ перехода	Комментарий
1		запуск системы	∅	

Класс объекта: cl_appication

Модификатор доступа: public

Метод: bild_tree_objects()

Функционал: построение дерева объекта

Параметры: Нет

Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий
1		построение дерева объекта	∅	

Класс объекта: cl_2

Модификатор доступа: public

Метод: cl_2

Функционал: параметризованный конструтор cl_2 наследуется от cl_base

Параметры: string name,cl_base* parent

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Класс объекта: cl_3

Модификатор доступа: public

Метод: cl_3

Функционал: параметризованный конструтор cl_3 наследуется от cl_base

Параметры: string name,cl_base* parent

Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Класс объекта: cl_4

Модификатор доступа: public

Метод: cl_4

Функционал: параметризованный конструтор cl_4 наследуется от cl_base

Параметры: string name,cl_base* parent

Возвращаемое значение: нет

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Функция: main()

Функционал: основная программа

Параметры: Нет

Возвращаемое значение: ob_cl_application.exec_app()

№	Предикат	Действия	№ перехода	Комментарий
1		возвращает функцию запуск системы	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_status()

Функционал: вводить номер исходного состояния очередного объекта

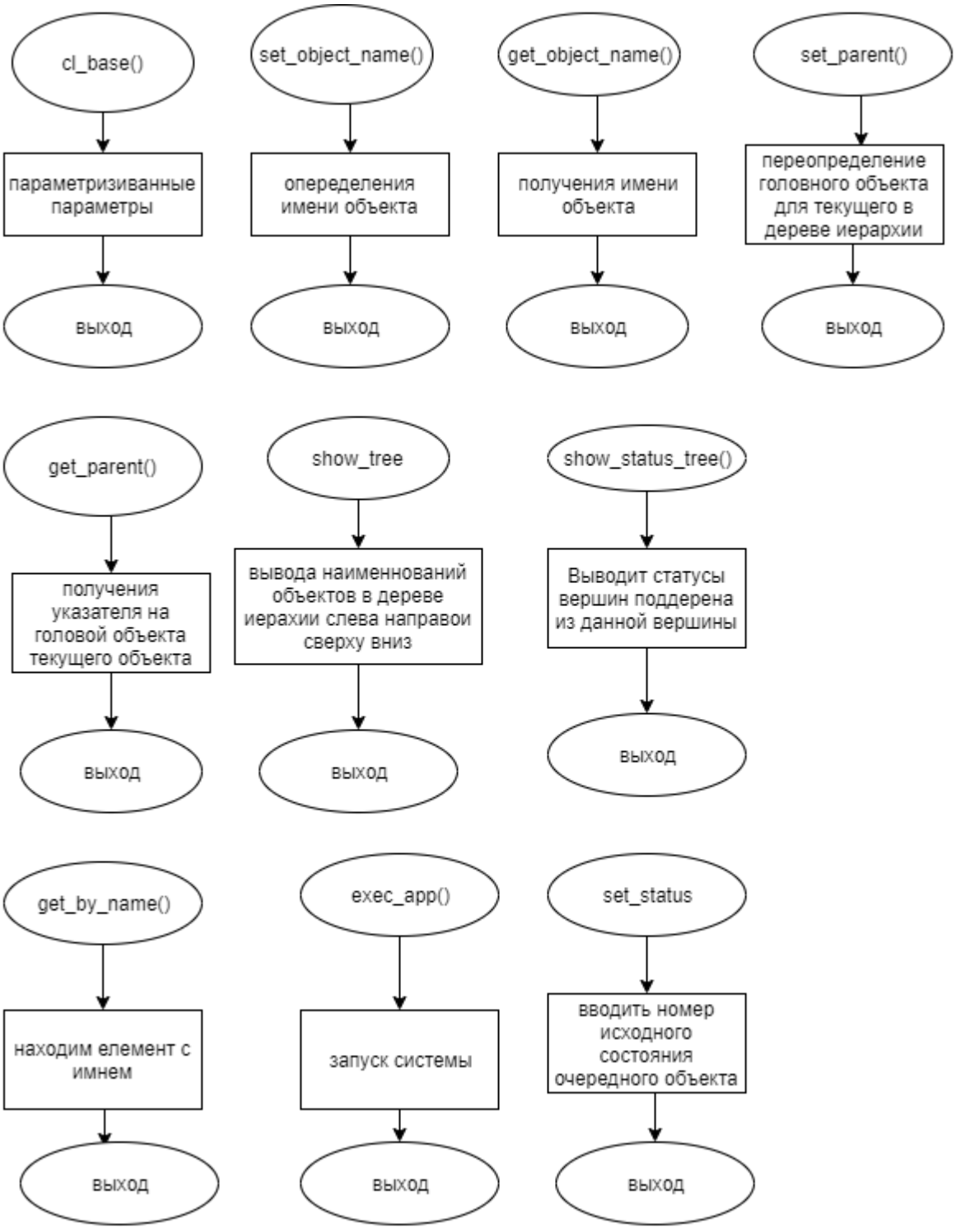
Параметры: status

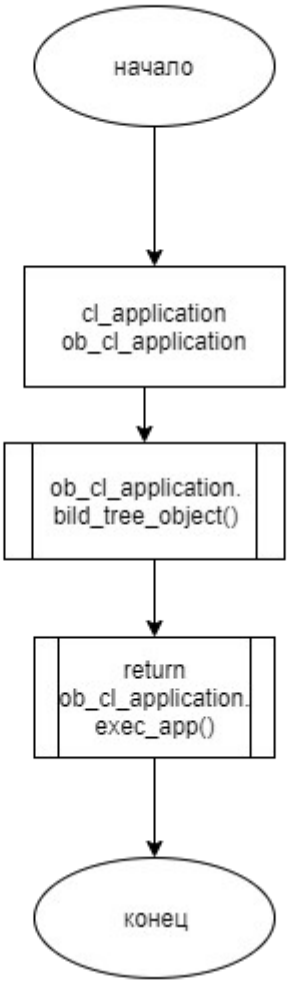
Возвращаемое значение: Нет

№	Предикат	Действия	№ перехода	Комментарий

1		вводить номер исходного состояния очередного объекта	∅	

Блок-схема алгоритма


cl_base()
параметризиванные параметры
выход
set_object_name()
опередения имени объекта
выход
get_object_name()
получения имени объекта
выход
set_parent()
переопределение головного объекта для текущего в дереве иерархии
выход
get_parent()
получения указателя на головой объекта текущего объекта
выход
show_tree
вывода наименнований объектов в дереве иерахии слева направои сверху вниз
выход
show_status_tree()
Выводит статусы вершин поддерена из данной вершины
выход
get_by_name()
находим елемент с имнем
выход
exec_app()
запуск системы
выход
set_status
вводить номер исходного состояния очередного объекта
выход



Код программы

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_application.cpp

```
#include<iostream>
#include "cl_application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_base.h"
using namespace std;
void cl_application::bild_tree_objects()
{
        cl_2* ob_2;
        cl_3* ob_3;
        cl_4* ob_4;
        string nameParent,nameChild;
        int class_type = 0,status=1;
        cin>>nameParent;
        this->set_object_name(nameParent);
        while(1){
                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_2=new cl_2(get_by_name(nameParent),nameChild);
                        ob_2->set_status(status);
                }
                else if(class_type==3)
                {
                        ob_3=new cl_3(get_by_name(nameParent),nameChild);
                        ob_3->set_status(status);
                }
                else if(class_type==4)
                {
                        ob_4=new cl_4(get_by_name(nameParent),nameChild);
                        ob_4->set_status(status);
                }
        }
}
int cl_application::exec_app()
{
        cout<<"Test result"<<endl;
        this->show_status_tree();
        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;
```

```
}
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();
        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 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