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

Новые классы:

Классы	модификатор доступа	Описание
A	public	Класс подчиненных объектов корневому объекту класса Application
B	public	Класс подчиненных объектов корневому объекту класса Application
C	public	Класс подчиненных объектов корневому объекту класса Application

в классе Base добавлены новые методы:

- void set_state(int) - метод определения состояния готовности объекта

- int get_state() - метод получения состояния объекта

- void show_obj_state() - метод вывода готовности объектов

- Base* object_search - метод поиска объекта по имени

добавлено новое свойство status - состояние готовности объекта

в классе Application добавлен новый метод:

- void build_new() - новый метод построения иерархического дерева

Локальные переменные метода build_new():

string s1, s2 - наименования головного и подчиненного объектов

int cl_type - номер принадлежности объекта

int obj_s - состояние объекта

Base* b, child - указатели на головной и подчиненный объекты

Описание алгоритма

Класс объекта: Base

Модификатор доступа: public

Метод: set_state

Функционал: определение состояния текущего объекта

Параметры: int status - состояние объекта

Возвращаемое значение: -

№	Предикат	Действия	№ перехода	Комментарий
1		присвоение статусу текущего объекта status	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: get_state

Функционал: получение состояния готовности текущего объекта

Параметры: -

Возвращаемое значение: int status - состояние текущего объекта

№	Предикат	Действия	№ перехода	Комментарий
1		возврат поля status текущего объекта	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: show_obj_state

Функционал: вывод состояния готовности каждого объекта

Параметры: -

Возвращаемое значение: -

№	Предикат	Действия	№ перехода	Комментарий
1	состояние текущего объекта положительное	вывод имени объекта и строки " is ready"	2	
		вывод имени объекта и строки	2	

		" is not ready"		
2	у текущего объекта есть подчиненные	перенос строки	3	
			3	
3	все подчиненные объекты обработаны		5	
			4	
4		повторный вызов метода для текущего подчиненного объекта	1	
5	текущий объект не равен последнему	перенос строки	∅	
			∅	

Класс объекта: Base

Модификатор доступа: public

Метод: get_by_name

Функционал: поиск объекта по имени

Параметры: string obj_name - наименование текущего объекта

Возвращаемое значение: string - наименование объекта

№	Предикат	Действия	№ перехода	Комментарий
1	Имя объекта равно obj_name	возврат этого объекта	∅	
			2	
2	все подчиненные объекты обработаны		4	
		повторный вызов метода для текущего подчиненного объекта с параметром obj_name	3	
3	b не равно 0	возврат b	∅	
			2	
4		возврат нулевого указателя	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: new_build

Функционал: строит новое иерархическое дерево из введенных данных

Параметры: -

Возвращаемое значение: -

№	Предикат	Действия	№ перехода	Комментарий
1		инициализация s1, s2 - наименований объектов	2	
2		инициализация cl_type, obj_s - номер принадлежности и состояние объекта	3	
3		считывание имени s1	4	
4		вызов методов set_name и set_state у данного объекта с аргументами s1 и 1 соответсвенно	5	
5		инициализация b, child -указателей на головной и подчиненный объекты	6	
6		считывание s1	7	
7	s1 не равно "endtree"		8	
			∅	
8		считывание s2, cl_type, obj_s	9	
9	b существует		10	
			11	
10	имя объекта равно текущему имени s1	присваивание полю child текущий указатель	11	
		вызов метода get_by_name и присваивание результата переменной child	11	
11	cl_type равен 2	создание объекта b класса A с параметрами child и s2	14	
			12	
12	cl_type равен 3	создание объекта b класса B с параметрами child и s2	14	
			13	
13		продолжаем строить дерево	6	
14		вызов метода set_state текущему подчиненному объекту с параметром obj_s	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: exec_app

Функционал: запуск системы

Параметры: -

Возвращаемое значение: int - код выполнения программы

№	Предикат	Действия	№ перехода	Комментарий
1		вывод строки "Test result"	2	
2		вызов метода show_obj_state()	3	
3		возврат 0 - кода выполнения программы	∅	

Класс объекта: A

Модификатор доступа: public

Метод: A

Функционал: параметризированный конструктор, определяет головной объект и наименование для текущего

Параметры: Base* obj_parent, string obj_name - указатель на объект и его имя

Возвращаемое значение: -

№	Предикат	Действия	№ перехода	Комментарий
1		вызов конструктора базового класса Base с параметрами obj_parent и obj_name	∅	

Класс объекта: B

Модификатор доступа: public

Метод: B

Функционал: параметризированный конструктор, определяет головной объект и наименование для текущего

Параметры: Base* obj_parent, string obj_name - указатель на объект и его имя

Возвращаемое значение: -

№	Предикат	Действия	№ перехода	Комментарий
1		вызов конструктора базового класса Base с параметрами obj_parent и obj_name	∅	

Класс объекта: C

Модификатор доступа: public

Метод: C

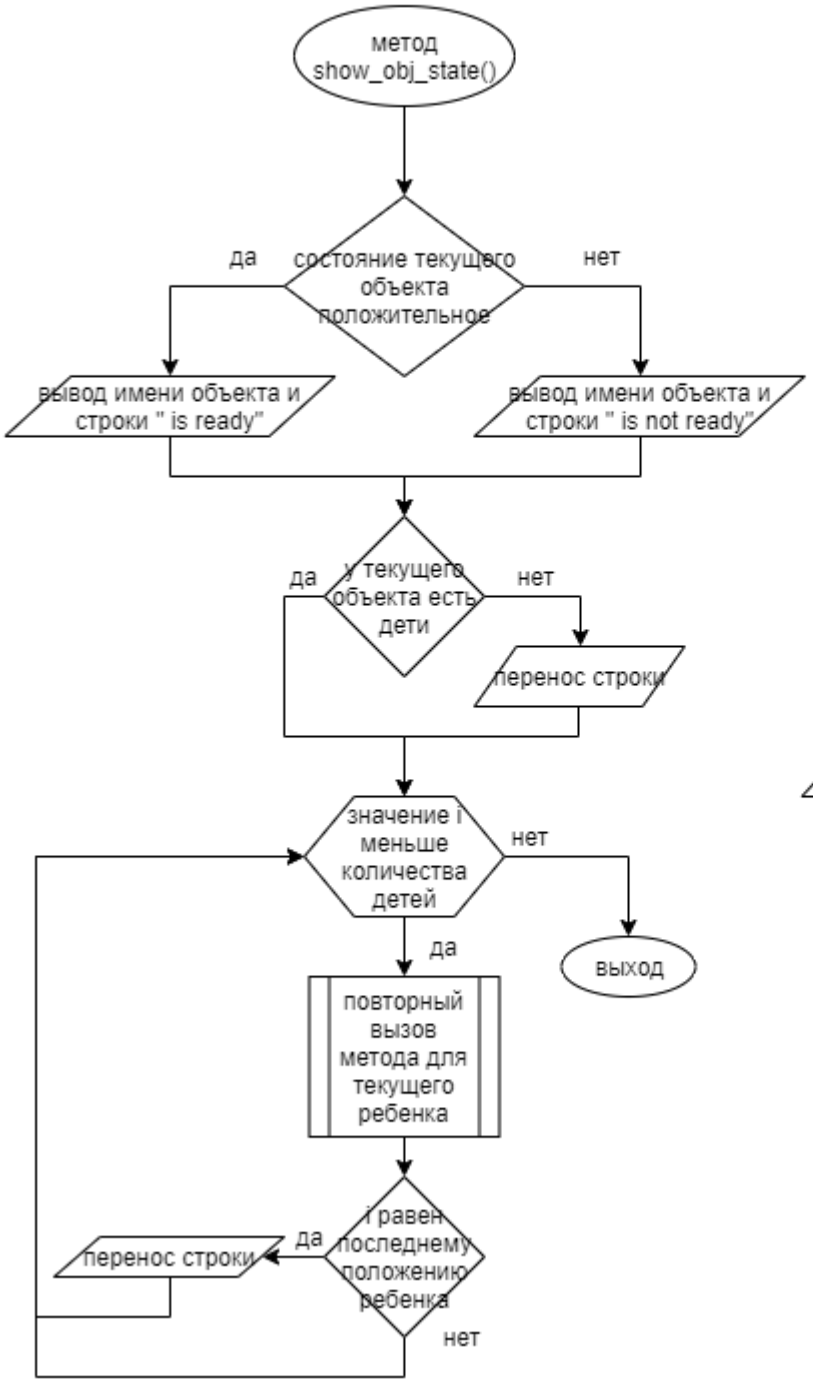
Функционал: параметризированный конструктор, определяет головной объект и наименование для текущего

Параметры: Base* obj_parent, string obj_name - указатель на объект и его имя

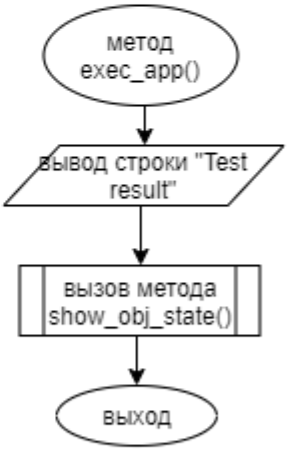
Возвращаемое значение: -

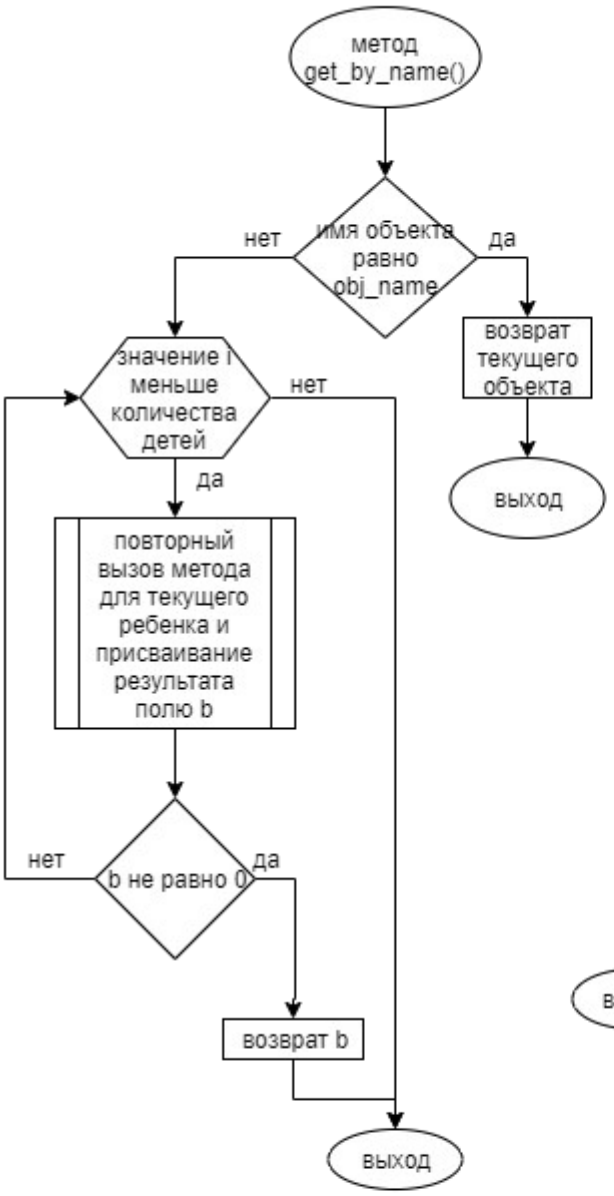
№	Предикат	Действия	№ перехода	Комментарий
1		вызов конструктора базового класса Base с параметрами obj_parent и obj_name	∅	

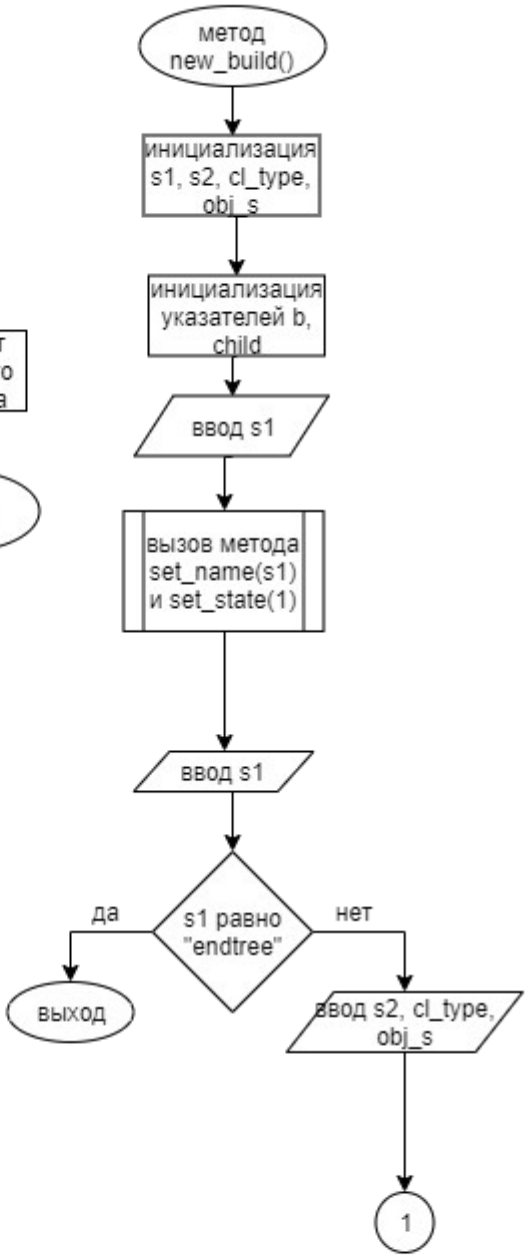
Блок-схема алгоритма

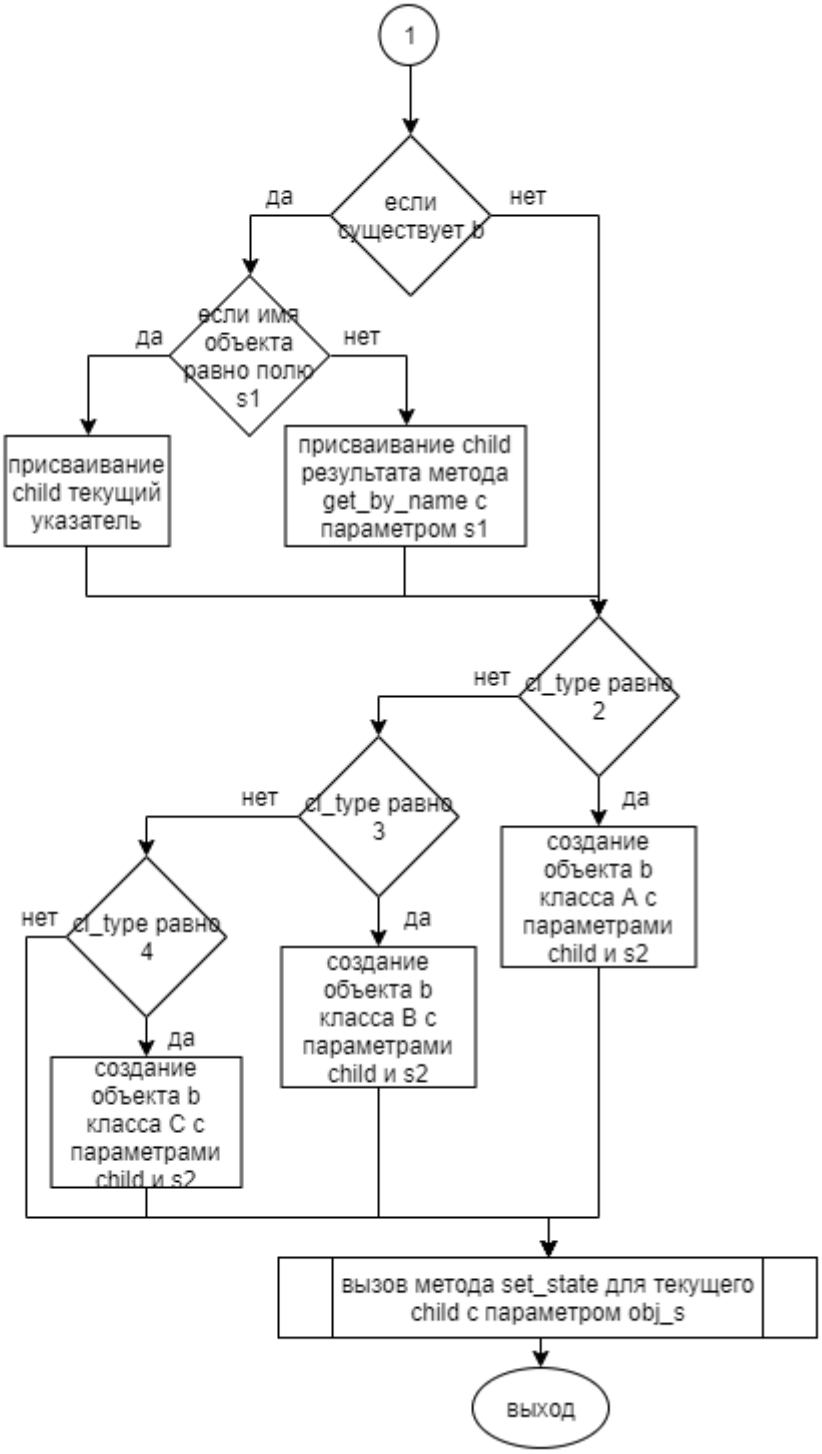

метод show_obj_state()
состояние текущего объекта положительное
да
нет
вывод имени объекта и строки " is ready"
вывод имени объекта и строки " is not ready"
у текущего объекта есть дети
да
нет
перенос строки
значение i меньше количества детей
нет
выход
да
повторный вызов метода для текущего ребенка
i равен последнему положению ребенка
да
перенос строки
нет

метод set_state(int)
присваивание полю status текущее значение status
конец


метод exec_app()
вывод строки "Test result"
вызов метода show_obj_state()
выход


метод get_by_name()
имя объекта равно obj_name
нет
да
возврат текущего объекта
выход
значение i меньше количества детей
нет
да
повторный вызов метода для текущего ребенка и присваивание результата полю b
b не равно 0
нет
да
возврат b
выход


метод new_build()
инициализация s1, s2, cl_type, obj_s
инициализация указателей b, child
ввод s1
вызов метода set_name(s1) и set_state(1)
ввод s1
s1 равно "endtree"
да
нет
выход
ввод s2, cl_type, obj_s
1



Код программы

Файл A.cpp

```
#include "A.h"
A::A(Base* obj_parent, string obj_name):Base(obj_parent, obj_name) {};
```

Файл A.h

```
#ifndef A_H
#define A_H
#include "Base.h"
class A:public Base
{
public:
        A(Base* obj_parent, string obj_name);
};
#endif
```

Файл Application.cpp

```
#include "Application.h"
#include "A.h"
#include "B.h"
#include "C.h"
#include <iostream>
using namespace std;

Application::Application(Base* obj_parent):Base(obj_parent, "") {};

void Application::new_build()
{
        string s1, s2;
        int cl_type = 0, obj_s;
        cin >> s1;
        set_name(s1);
        set_state(1);

        Base* b = nullptr;
        Base* child = this;
        while (true)
        {
                cin >> s1;
                if (s1 == "endtree") break;
                cin >> s2 >> cl_type >> obj_s;
```

```
                if(b)
                {
                        if(this->get_name() == s1) child = this;
                        else child = get_by_name(s1);
                }
                if(cl_type == 2) b = new A(child, s2);
                else if(cl_type == 3) b = new B(child, s2);
                else if(cl_type == 4) b = new C(child, s2);
                else continue;

                b->set_state(obj_s);
        }
};

void Application::build_tree()
{
        Base* lead_parent; // Указатель на головной объект
        Base* child_parent; // Указатель на дочерний объект
        string l_prt, c_prt;
        cin >> l_prt;
        set_name(l_prt);
        lead_parent = this;

        while (true)
        {
                cin >> l_prt >> c_prt;
                if (l_prt == c_prt) return;

                if (l_prt != lead_parent->get_name())
                {
                        if (l_prt == child_parent->get_name())
                        {
                                lead_parent = child_parent;
                        }
                        else continue;
                }
                child_parent = new A(lead_parent, c_prt);
        }
};

int Application::exec_app()
{
        cout << "Test result\n";
        show_obj_state();
        return 0;
};
```

Файл Application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H
#include "Base.h"

class Application:public Base
```

```
{
public:
        Application(Base* obj_parent = 0);
        void build_tree(); // Метод построения дерева
        int exec_app(); // Метод запуска системы
        void new_build(); // Метод построения иерархического дерева
};
#endif
```

Файл Base.cpp

```
#include "Base.h"
#include <iostream>
using namespace std;

Base::Base(Base* obj_parent, string obj_name)
{
        set_parent(obj_parent);
        set_name(obj_name);
};
Base::~Base()
{
        for(int i = 0; i < children.size(); i++) delete children[i];
};
void Base::set_name(string obj_name)
{
        this->name = obj_name;
};
string Base::get_name()
{
        return this->name;
};
void Base::set_parent(Base* obj_parent)
{
        if(obj_parent)
        {
                if(this->parent)
                {
                        for(int i = 0; i < this->parent->children.size(); i++)
                        {
                                if(this == this->parent->children[i])
                                {
                                        this->parent-
>children.erase(children.begin() + i);
                                }
                        }
                }
                this->parent = obj_parent;
                this->parent->children.push_back(this);
        }
```

```
        else this->parent = 0;
};
Base* Base::get_parent()
{
        return this->parent;
};
void Base::print_objects()
{
        vector <Base*>::iterator c_iterator;
        if (children.empty()) return;
        cout << endl << name;

        c_iterator = children.begin();
        while (c_iterator != children.end()) {
                cout << "  " << (*c_iterator)->get_name();
                c_iterator++;
        }
        c_iterator--;
        (*c_iterator)->print_objects();
};
void Base::set_state(int status)
{
        this->status = status;
};
int Base::get_state()
{
        return status;
};
void Base::show_obj_state()
{
        cout << "The object " << this->name;
        if (this->status > 0) cout << " is ready";
        else if (this->status <= 0) cout << " is not ready";

        if (this->children.size()) cout << "\n";
        for (int i = 0; i < this->children.size(); ++i)
        {
                this->children[i]->show_obj_state();
                if (i != this->children.size() - 1) cout << "\n";
        }
        return;
};
Base* Base::get_by_name(string obj_name) {
        if (this->name == obj_name) {
        return this;
};
        for (int i = 0; i < this->children.size(); ++i)
        {
                Base* b = children[i]->get_by_name(obj_name);
                if (b != NULL) return b;
        }
        return nullptr;
}
```

Файл Base.h

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

class Base
{
        string name; // Имя объекта
        Base* parent = nullptr; // Указатель на головной объект
        vector <Base*> children; // Массив подчиненных объектов
        int status; // Состояние объекта

public:
        Base(Base*, string = "root"); // Конструктор класса Base
        ~Base();
        void set_name(string); // Определение имени объекта
        string get_name(); // Получение имени объекта

        void set_parent(Base*); // Переопределение указателя головного объекта
        Base* get_parent(); // Получение указателя на головной объект

        void print_objects(); // Вывод имен оъектов

        void set_state(int); // Определение состояния объекта
        int get_state(); // Получение состояния объекта
        void show_obj_state(); // Вывод состояния готовности объектов
        Base* get_by_name(string); // Поиск объектов по имени
};
#endif
```

Файл B.cpp

```
#include "B.h"
B::B(Base* obj_parent, string obj_name):Base(obj_parent, obj_name) {};
```

Файл B.h

```
#ifndef B_H
#define B_H
#include "Base.h"
```

```
class B:public Base
{
public:
        B(Base* obj_parent, string obj_name);
};
#endif
```

Файл C.cpp

```
#include "C.h"
C::C(Base* obj_parent, string obj_name):Base(obj_parent, obj_name) {};
```

Файл C.h

```
#ifndef C_H
#define C_H
#include "Base.h"
class C:public Base
{
public:
        C(Base* obj_parent, string obj_name);
};
#endif
```

Файл main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "Application.h"

int main()
{
        Application app(nullptr);
        app.new_build();  // построение дерева объектов
        return app.exec_app();    // запуск системы
}
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root root ob1 2 1 root ob2 2 -2 root ob3 3 1 ob2 ob4 2 1 ob1 ob5 4 2 ob5 ob6 2 -1 endtree	Test result The object root is ready The object ob1 is ready The object ob5 is ready The object ob6 is not ready The object ob2 is not ready The object ob4 is ready The object ob3 is ready	Test result The object root is ready The object ob1 is ready The object ob5 is ready The object ob6 is not ready The object ob2 is not ready The object ob4 is ready The object ob3 is ready
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