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	ИВБО-04-20	Пивоваров С.С.
Руководитель практики	Ассистент	Данилович Е.С.
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

class BasicClass

поля класса:

- state - int, состояние объекта

публичные методы класса:

- setState(int) - возвращаемый тип void, устанавливает состояние объекта
- getState() - возвращаемый тип int, возвращает состояние объекта
- searchObject(string) - возвращаемый тип BasicClass*, возвращает указатель по имени
- printReadyObj() - возвращаемый тип void, вывод деревва иерархии и состояний объектов

class MyApp

приватные методы класса:

- addNewObj(BasicClass*, string, int, int) - возвращаемый void, распределяет создание объектов в зависимости от номера класса

публичные методы класса:

- MyApp(BasicClass*, string) - конструктор с параметрами по умолчанию, делигирование конструктору базового класса с параметрами
- build_tree_object() - возвращаемый тип void, построение дерева иерархии
- exec_app() - возвращаемый тип int, вывод дерева иерархии

class MyClass2

публичные методы:

- MyClass2(BasicClass*, string, int) - конструктор с параметрами, делигирование конструктору базового класса с параметрами

class MyClass3

публичные методы:

- MyClass3(BasicClass*, string, int) - конструктор с параметрами, делигирование конструктору базового класса с парматерами

class MyClass4

- MyClass4(BasicClass*, string, int) - конструктор с параметрами, делигирование конструктору базового класса с параметрами

Описание алгоритма

Функция: main()

Функционал: создание приложения и построение дерева иерархии

Параметры: -

Возвращаемое значение: int, код возврата

№	Предикат	Действия	№ перехода	Комментарий
1		создание объекта приложения app класса MyApp	2	
2		вызов метода построения дерева иерархии build_tree()	3	
3		вызов метода для вывода дерева иерархии в консоль exec_app()	∅	

Класс объекта: BasicClass

Модификатор доступа: public

Метод: setState()

Функционал: инициализация состояния объекта

Параметры: int - состояние объекта

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		присвоение переменной state значения state	∅	

Класс объекта: BasicClass

Модификатор доступа: public

Метод: getState()

Функционал: возврат состояния объекта

Параметры: -

Возвращаемое значение: int - состояние объекта

№	Предикат	Действия	№ перехода	Комментарий
1		возврат state	∅	

Класс объекта: BasicClass

Модификатор доступа: public

Метод: searchObject()

Функционал: поиск указателя на объект по его имени

Параметры: string - имя искомого объекта

Возвращаемое значение: BasicClass*

№	Предикат	Действия	№ перехода	Комментарий
1		инициализация переменной типа int, i = 0	2	
2	i < количества детей		3	
			4	
3	name == имени i-того ребенка	возврат i-того указателя на объект	∅	
		i++	2	
4		инициализация переменной типа int, i = 0	5	
5	i < количества детей		6	
		возврат указателя nullptr	∅	
6	указатель на ребенка != nullptr	возврат результата вызова метода searchObject(name) у i-того объекта	∅	
		i++	4	

Класс объекта: BasicClass

Модификатор доступа: public

Метод: printReadyObject()

Функционал: вывод в консоль дерева объектов

Параметры: -

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1	количество детей > 0		2	
			∅	
2		иницализация переменной типа, int i = 0	3	
3	i < количества детей	вывод "\nThe object ", children[i].getName()	4	
			∅	
4	состояние i-того ребенка > 0	вывод " is ready"	5	
		вывод " is not ready"	5	
5	количество детей у i-того объекта > 0	вызов метода printReadyObj() у i-того объекта	6	
			6	
6		i++	3	

Класс объекта: MyApp

Модификатор доступа: private

Метод: addNewObj()

Функционал: создание нового объекта в зависимости от указанного номера класса

Параметры: BasicClass* - родитель, string - имя, int - номер класса, int - сосояние

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1	number == 2	создание новог объекта MyClass2(parent, nameNewObj, state)	∅	
			2	
2	number == 3	создание новог объекта MyClass3(parent, nameNewObj, state)	∅	
		создание новог объекта MyClass4(parent, nameNewObj, state)	∅	

Класс объекта: MyApp

Модификатор доступа: public

Метод: MyApp()

Функционал: конструктор

Параметры: BasicClass* - родитель, string - имя

Возвращаемое значение: конструктор

№	Предикат	Действия	№ перехода	Комментарий
1		делигирование вызова конструктора базовго класса с параметрами	2	
2		вызов метода setState(1)	∅	

Класс объекта: MyApp

Модификатор доступа: public

Метод: build_tree()

Функционал: построение дерева иерархии

Параметры: -

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		объявляем переменную tempNametипа типа string	2	
2		ввод tempName	3	
3		вызов метода setName(tempName)	4	
4		создание контейнеров pair: intPair типа int, int strPair типа string, string инициализируем указатель anyPtr типа BasicClass* со значением nullptr	5	
5	true		6	
			∅	
6		ввод strPair.first	7	
7	первое слово == "endtree"		∅	
			8	
8		ввод strPair.second, intPair.first, intPair.second	9	
9	первое слово == имя приложения	вызов метода addNewObj(this, strPair.second, intPair.first, intPair.second)	5	

		присвоение указателю anyPtr результата вызова метода searchObject(str.first) вызов метода addNewObj(anyPtr, strPair.second, intPair.first, intPair.second)	5	

Класс объекта: MyApp

Модификатор доступа: public

Метод: exec_app()

Функционал: вывод дерева объекта в консоль

Параметры: -

Возвращаемое значение: int - код возврата

№	Предикат	Действия	№ перехода	Комментарий
1		вывод "\nThe object ", this->getName(), " is ready"	2	
2		вызов метода printReadyObj();	3	
3		возврат 0	∅	

Класс объекта: MyClass2

Модификатор доступа: public

Метод: MyClass2

Функционал: конструктор

Параметры: BasicClass* - родитель, string - имя, int - состояние объекта

Возвращаемое значение: конструктор

№	Предикат	Действия	№ перехода	Комментарий
1		делигирование вызова конструктора базового класса BasicClass(parent, name)	2	
2		вызов метода setState(state)	∅	

Класс объекта: MyClass3

Модификатор доступа: public

Метод: MyClass3

Функционал: конструктор

Параметры: BasicClass* - родитель, string - имя, int - состояние объекта

Возвращаемое значение: конструктор

№	Предикат	Действия	№ перехода	Комментарий
1		делигирование вызова конструктора базового класса BasicClass(parent, name)	2	
2		вызов метода setState(state)	∅	

Класс объекта: MyClasst4

Модификатор доступа: public

Метод: MyClass4()

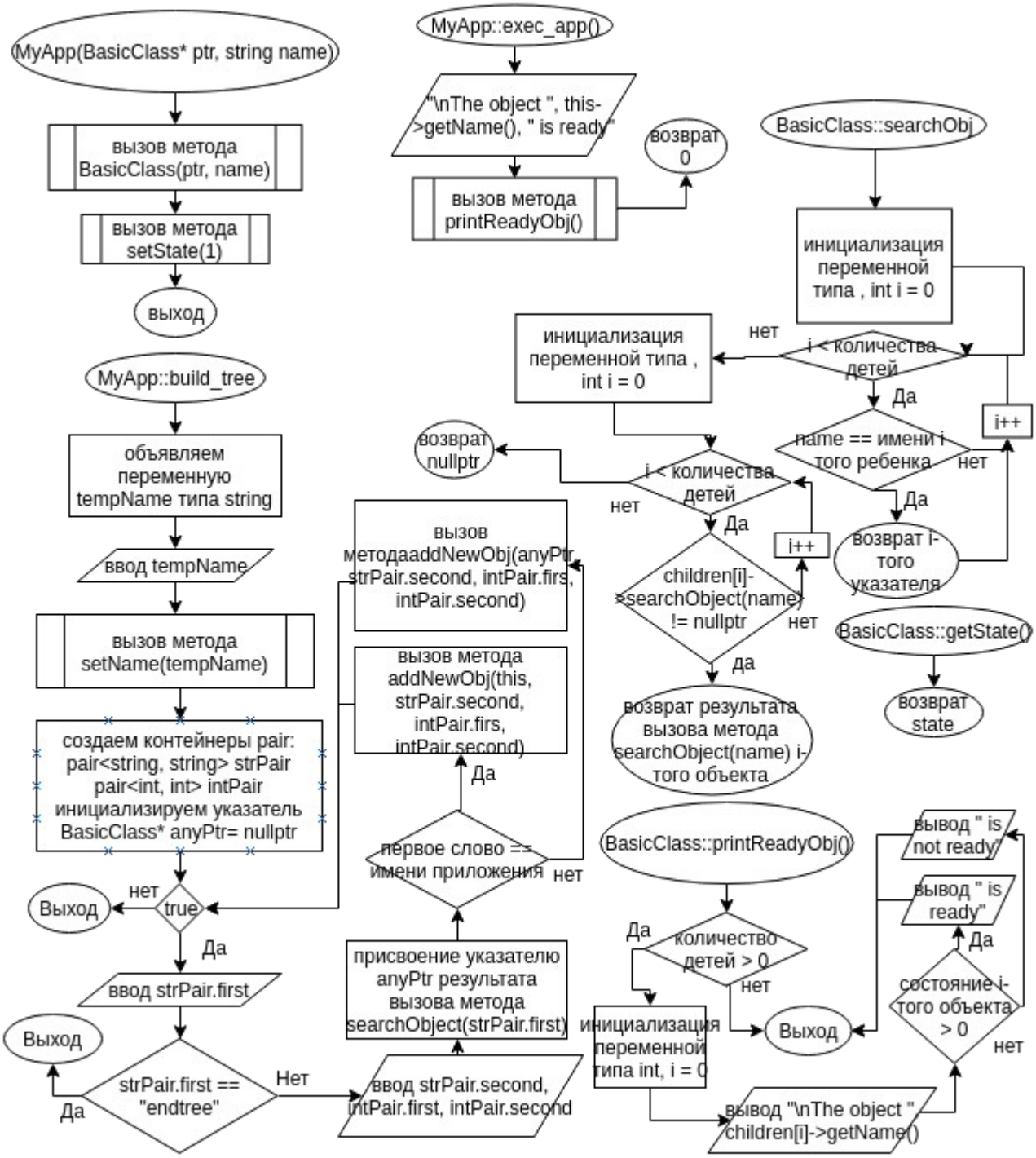
Функционал: конструктор

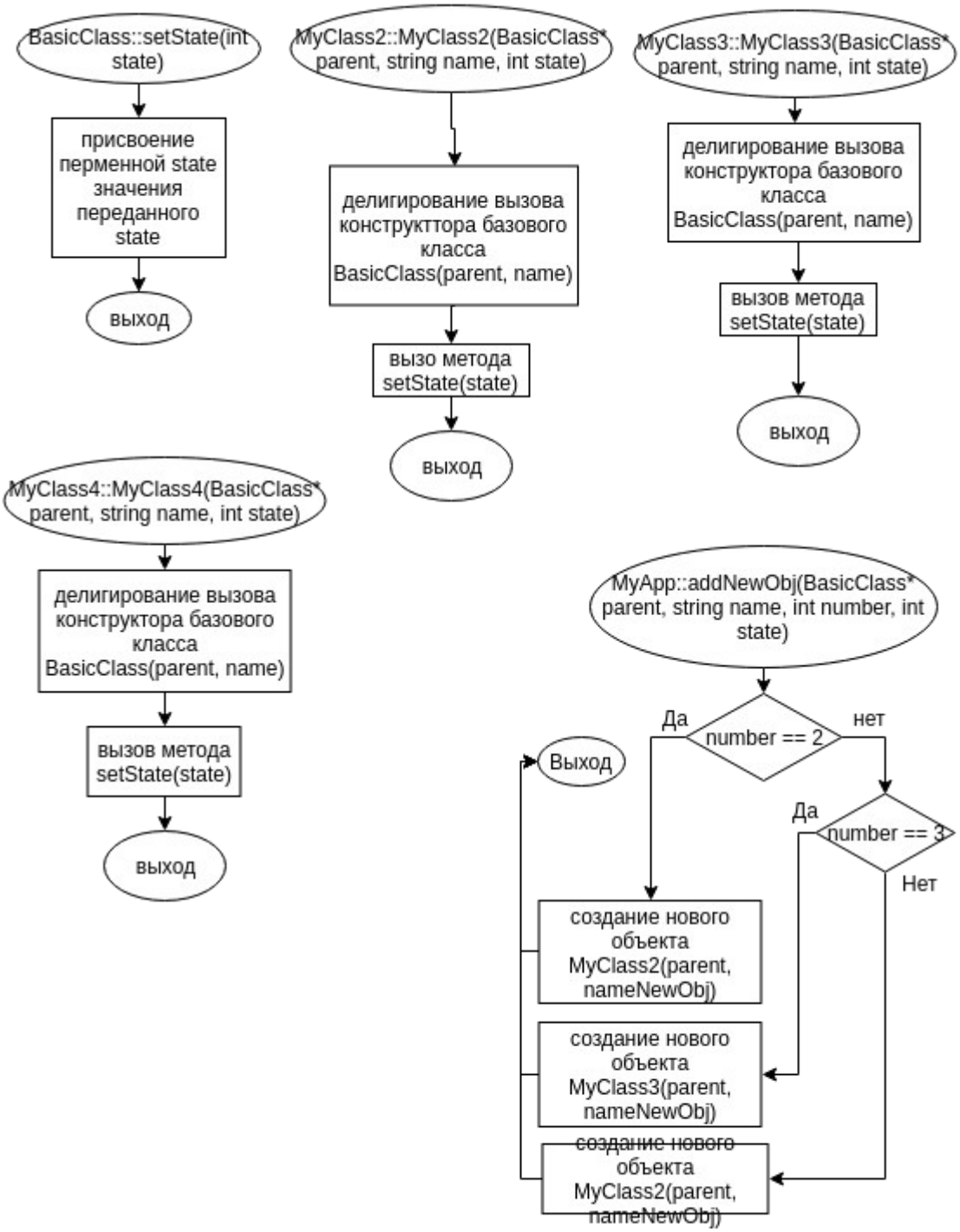
Параметры: BasicClass* - родитель, string - имя, int - состояние объекта

Возвращаемое значение: конструктор

№	Предикат	Действия	№ перехода	Комментарий
1		делигирование вызова конструктруктора базового класса BasicClass(paren, name)	2	
2		вызов метода setState(state)	∅	

Блок-схема алгоритма


MyApp(BasicClass* ptr, string name)
вызов метода BasicClass(ptr, name)
вызов метода setState(1)
выход
MyApp::build_tree
объявляем переменную tempName типа string
ввод tempName
вызов метода setName(tempName)
создаем контейнеры pair: pair<string, string> strPair pair<int, int> intPair инициализируем указатель BasicClass* anyPtr= nullptr
Выход
нет
true
Да
ввод strPair.first
Выход
strPair.first == "endtree"
Да
Нет
ввод strPair.second, intPair.first, intPair.second
присвоение указателю anyPtr результата вызова метода searchObject(strPair.first)
первое слово == имени приложения
нет
Да
вызов методаaddNewObj(anyPtr strPair.second, intPair.firs, intPair.second)
вызов метода addNewObj(this, strPair.second, intPair.firs, intPair.second)
MyApp::exec_app()
"\nThe object ", this->getName(), " is ready"
вызов метода printReadyObj()
возврат 0
BasicClass::printReadyObj()
количество детей > 0
Да
нет
инициализация переменной типа int, i = 0
Выход
вывод "\nThe object " children[i]->getName()
состояние i-того объекта > 0
Да
нет
вывод " is ready"
вывод " is not ready"
BasicClass::searchObj
инициализация переменной типа , int i = 0
i < количества детей
нет
Да
name == имени i того ребенка
Да
нет
i++
возврат i-того указателя
инициализация переменной типа , int i = 0
i < количества детей
нет
Да
возврат nullptr
children[i]->searchObject(name) != nullptr
нет
да
i++
возврат результата вызова метода searchObject(name) i-того объекта
BasicClass::getState()
возврат state



Код программы

Файл basicClass.cpp

```
#include "basicClass.h"

BasicClass::BasicClass(BasicClass* ptr, string name) : ptrParent(ptr),
name(name)
{
        if (ptr != nullptr)
                ptr->children.push_back(this);
}

void BasicClass::setName(string name)
{
        this->name = name;
}

void BasicClass::setState(int state) {
        this->state = state;
}

string BasicClass::getName()
{
        return name;
}

int BasicClass::getState() {
        return state;
}

BasicClass* BasicClass::getParent()
{
        return ptrParent;
}

void BasicClass::setParent(BasicClass* parent)
{
        this->ptrParent = parent;
}

void BasicClass::printObjects()
{
        cout << "\n" << name;
        if (children.size() > 0)
                for(int i = 0; i < children.size(); i++)
                        cout << "  " << children[i]->getName();
                if (children[children.size()-1]->children.size() > 0)
                        children[children.size()-1]->printObjects();
}

void BasicClass::printReadyObj() {
        if (children.size() > 0)
                for (int i = 0; i < children.size(); i++) {
                        cout << "\nThe object " << children[i]->getName();
                        if (children[i]->getState() > 0)
                                cout << " is ready";
```

```
                        else
                                cout << " is not ready";
                        if (children[i]->children.size()>0)
                                children[i]->printReadyObj();
                }
}

BasicClass* BasicClass::searchObject(string name) {
        for (int i = 0; i < children.size(); i++) {
                if (name == children[i]->getName())
                        return children[i];
        }

        for (int i = 0; i < children.size(); i++) {
                if (children[i]->searchObject(name )!= nullptr)
                        return children[i]->searchObject(name);
        }

        return nullptr;
}

BasicClass::~BasicClass()
{
        for(int i = 0; i < children.size(); i++)
                delete children[i];
}
```

Файл basicClass.h

```
#ifndef BASICCLASS_H
#define BASICCLASS_H

#include <string>
#include <iostream>
#include <vector>

using namespace std;

class BasicClass
{
private:

        string name;
        BasicClass* ptrParent;
        vector<BasicClass*> children;
        int state;

public:

        BasicClass(BasicClass* ptr = nullptr, string name = "Object_root");
        void setName(string name);
        void setState(int);
        string getName();
        int getState();
```

```
        void printObjects();
        void printReadyObj();
        void setParent(BasicClass*);
        BasicClass* getParent();
        BasicClass* searchObject(string name);
        ~BasicClass();

};

#endif
```

Файл class2.cpp

```
#include "class2.h"

MyClass2::MyClass2(BasicClass* parent, string name, int state) :
BasicClass(parent, name) {
        setState(state);
}
```

Файл class2.h

```
#ifndef CLASS2_H
#define CLASS2_H

#include "basicClass.h"

class MyClass2 : public BasicClass
{
public:
        MyClass2(BasicClass* parent, string name, int);
};

#endif
```

Файл class3.cpp

```
#include "class3.h"

MyClass3::MyClass3(BasicClass* parent, string name, int state) :
BasicClass(parent, name) {
        setState(state);
}
```

Файл class3.h

```
#ifndef CLASS3_H
#define CLASS3_H

#include "basicClass.h"

class MyClass3 : public BasicClass
{
public:
        MyClass3(BasicClass* parent, string name, int);
};

#endif
```

Файл class4.cpp

```
#include "class4.h"

MyClass4::MyClass4(BasicClass* parent, string name, int state) :
BasicClass(parent, name) {
        setState(state);
}
```

Файл class4.h

```
#ifndef CLASS4_H
#define CLASS4_H

#include "basicClass.h"

class MyClass4 : public BasicClass
{
public:
        MyClass4(BasicClass* parent, string name, int);
```

```
};

#endif
```

Файл main.cpp

```
#include "MyApp.h"

int main()
{
	MyApp app(nullptr);
	app.bild_tree_object();
	return app.exec_app();
}
```

Файл MyApp.cpp

```
#include "MyApp.h"

MyApp::MyApp(BasicClass* ptr, string name) : BasicClass(ptr, name) {}

void MyApp::bild_tree_object()
{
	string tempName;
	cin >> tempName;
	setName(tempName);

	pair<string, string> strPair;
	pair<int, int> intPair;

	BasicClass* anyPtr = nullptr;

	while(true) {
		cin >> strPair.first;

		if (strPair.first == "endtree")
			break;

		cin >> strPair.second >> intPair.first >> intPair.second;

		if (strPair.first == this->getName())
			addNewObj(this, strPair.second, intPair.first,
intPair.second);
		else {
```

```
                        anyPtr = searchObject(strPair.first);
                        addNewObj(anyPtr, strPair.second, intPair.first,
intPair.second);
                }
        }
}

void MyApp::addNewObj(BasicClass* parent, string nameNewObj, int number, int
state) {
        if (number == 2)
                new MyClass2(parent, nameNewObj, state);
        else if (number == 3)
                new MyClass3(parent, nameNewObj, state);
        else
                new MyClass4(parent, nameNewObj, state);
}

int MyApp::exec_app()
{
        cout << "Test result\nThe object " <<  this->getName() << " is ready";
        printReadyObj();
        return 0;
}
```

Файл MyApp.h

```
#ifndef MYAPP_H
#define MYAPP_H

#include "basicClass.h"
#include "class2.h"
#include "class3.h"
#include "class4.h"

class MyApp : public BasicClass
{
private:

        void addNewObj(BasicClass*, string, int, int);

public:

        MyApp(BasicClass* ptr = nullptr, string name = "app_root");
        void bild_tree_object();
        int exec_app();

};

#endif
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
putin putin medvedev 3 1 putin peskov 2 1 medvedev navalny 3 -1 medvedev lavrov 3 1 putin ktoto 3 1 endtree	Test result The object putin is ready The object medvedev is ready The object navalny is not ready The object lavrov is ready The object peskov is ready The object ktoto is ready	Test result The object putin is ready The object medvedev is ready The object navalny is not ready The object lavrov is ready The object peskov is ready The object ktoto is ready