МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« КЛ_3_3 Сигналы и обработчики »

С тудент группы	ИНБО-02-21	Кадыров А.Х.
Руководитель практики	Старший преподаватель	Грач Е.П.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Постановка задачи

Реализация сигналов и обработчиков

Для организации взаимодействия объектов вне схемы взаимосвязи используется механизм сигналов и обработчиков. Вместе с передачей сигнала еще передается определенное множество данных. Механизм сигналов и обработчиков реализует схему взаимодействия объектов один ко многим.

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).

Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:

1. Установления связи между сигналом текущего объекта и обработчиком целевого объекта;
2. Удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
3. Выдачи сигнала от текущего объекта с передачей строковой переменной.

Включенный объект может выдать или обработать сигнал.

Методу установки связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу удаления (разрыва) связи передать указатель на метод сигнала текущего объекта, указатель на целевой объект и указатель на метод обработчика целевого объекта.

Методу выдачи сигнала передать указатель на метод сигнала и строковую переменную. В данном методе реализовать алгоритм:

1. Вызов метода сигнала с передачей строковой переменной по ссылке.
2. Цикл по всем связям сигнал-обработчик текущего объекта.
2.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то вызвать метод обработчика очередного целевого объекта и передать в качестве аргумента строковую переменную по значению.
3. Конец цикла.

Для приведения указателя на метод сигнала и на метод обработчика использовать параметризированное макроопределение препроцессора.

В базовый класс добавить метод определения абсолютного пути до текущего объекта. Этот метод возвращает абсолютный путь текущего объекта.

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 3 курсовой работы. Система содержит объекты шести классов с номерами: 1,2,3,4,5,6. Классу корневого объекта соответствует номер 1. В каждом производном классе реализовать один метод сигнала и один метод обработчика. Каждый метод сигнала с новой строки выводит:

`Signal from «абсолютная координата объекта»`

Каждый метод сигнала добавляет переданной по параметру строке текста номер класса принадлежности текущего объекта по форме:

`«пробел»(class: «номер класса»)`

Каждый метод обработчика с новой строки выводит:

`Signal to «абсолютная координата объекта» Text: «переданная строка»`

Реализовать алгоритм работы системы:

1. В методе построения системы:

1.1. Построение дерева иерархии объектов согласно вводу.

1.2. Ввод и построение множества связей сигнал-обработчик для заданных пар объектов.

2. В методе отработки системы:

2.1. Привести все объекты в состоянии готовности.

2.2. Цикл до признака завершения ввода.

2.2.1. Ввод наименования объекта и текста сообщения.

2.2.2. Вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.

2.3. Конец цикла.

Допускаем, что все входные данные вводятся синтаксически корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы.

Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.

Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве иерархии.

Структура данных для ввода согласно изложенному в версии № 3 курсовой работы.

После ввода состава дерева иерархии построчно вводится: «координата объекта выдающего сигнал»«координата целевого объекта»

Ввод информации для построения связей завершается строкой, которая

содержит

end_of_connections

В методе запуска (отработки) системы.

Построчно вводятся множество команд в производном порядке:

EMIT «координата объекта»«текст» - выдать сигнал от заданного по координате объекта;

SET_CONNECT «координата объекта выдающего сигнал»«координата целевого объекта» - установка связи;

DELETE_CONNECT «координата объекта выдающего сигнал»«координата целевого объекта» - удаление связи;

SET_CONDITION «координата объекта»«значение состояния» - установка состояния объекта.

END – завершить функционирование системы (выполнение программы).

Команда END присутствует обязательно.

Если координата объекта задана некорректно, то соответствующая операция не выполняется и с новой строки выдается сообщение об ошибке.

Если не найден объект по координате:

Object «координата объекта» not found

Если не найден целевой объект по координате:

Handler object «координата целевого объекта» not found

Пример ввода

appls_root

/ object_s1 3

```
/                object_s2                2
/object_s2             object_s4             4
/                object_s13                5
/object_s2             object_s6             6
/object_s1             object_s7             2
endtree
/object_s2/object_s4         /object_s2/object_s6
/object_s2              /object_s1/object_s7
/                   /object_s2/object_s4
/object_s2/object_s4                    /
end_of_connections
EMIT    /object_s2/object_s4    Send    message    1
DELETE_CONNECT        /object_s2/object_s4        /
EMIT    /object_s2/object_s4    Send    message    2
SET_CONDITION        /object_s2/object_s4        0
EMIT    /object_s2/object_s4    Send    message    3
SET_CONNECT     /object_s1     /object_s2/object_s6
EMIT      /object_s1      Send      message      4
END
```

Описание выходных данных

Первая строка:

`Object tree`

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

`Signal from «абсолютная координата объекта»`

Если отработал метод обработчика:

Signal to «абсолютная координата объекта»

Text: «переданная строка»

Пример вывода

Object tree

appls_root

object_s1

object_s7

object_s2

object_s4

object_s6

object_s13

Signal from /object_s2/object_s4

Signal to /object_s2/object_s6 Text: Send message 1 (class: 4)

Signal to / Text: Send message 1 (class: 4)

Signal from /object_s2/object_s4

Signal to /object_s2/object_s6 Text: Send message 2 (class: 4)

Signal from /object_s1

Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)

Метод решения

Класс FLevel:

Добавлены:

- void signal_1(string&) - метод сигнала;
- void handler_1(string) - метод приёмника.

Класс SLevel:

Добавлены:

- void signal_2(string&) - метод сигнала;
- void handler_2(string) - метод приёмника.

Класс TLevel:

Добавлены:

- void signal_3(string&) - метод сигнала;
- void handler_3(string) - метод приёмника.

Класс FoLevel:

Добавлены:

- void signal_4(string&) - метод сигнала;
- void handler_4(string) - метод приёмника.

Класс FifLevel:

Добавлены:

- void signal_5(string&) - метод сигнала;

- void handler_5(string) - метод приёмника.

Класс Tree:

Добавлены:

- void signal_0(string&) - метод сигнала;
- void handler_0(string) - метод приёмника.

Изменены:

- void buildTree() - добавлено считывание начальных связей объектов.

Класс Leaf:

Добавлены:

- string getAbsolutePath(Leaf* object) - получение полного пути объекта в иерархии объектов;
- void set_connect(SIGNAL signal, Leaf* target, HANDLER handler) = установление связи первого объекта со вторым;
- void delete_connect(SIGNAL signal, Leaf* target, HANDLER handler) = удаление связи первого объекта со вторым;
- void emit_signal(SIGNAL signal, string& command) = передача сообщения для всех соединений первого объекта;
- void setAllPrepaired(Leaf* leaf) = установка значения состояния данного параметра и всех его детей равным 1;
- void setAllNotPrepaired(Leaf* leaf) = установка значения состояния данного параметра и всех его детей равным 0;
- void readCommands(map<string, Leaf*> treeMap) = считывание команд;

- SIGNAL chooseSignal(int num) = выбор функции сигнала класса в зависимости от типа класса;
- HANDLER chooseHandler(int num) = выбор функции приёмника класса в зависимости от типа класса.

Описание алгоритма

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

Класс объекта: FLevel

Модификатор доступа: public

Метод: signal_1

Функционал: Функция сигнала

Параметры: string& command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 1.

Таблица 1. Алгоритм метода signal_1 класса FLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal from " + getAbsolutePath(this)	2	
2		K command прибавить " (class: 2)"	∅	

Класс объекта: FLevel

Модификатор доступа: public

Метод: handler_1

Функционал: Функция приёмника

Параметры: string command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода handler_1 класса FLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal to " + getAbsolutePath(this) + " Text: " + command	∅	

Класс объекта: TLevel

Модификатор доступа: public

Метод: handler_3

Функционал: Функция приёмника

Параметры: string command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода handler_3 класса TLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal to " + getAbsolutePath(this) + " Text: " + command	∅	

Класс объекта: FoLevel

Модификатор доступа: public

Метод: handler_4

Функционал: Функция приёмника

Параметры: string command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода handler_4 класса FoLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal to " + getAbsolutePath(this) + " Text: " + command	∅	

Класс объекта: FifLevel

Модификатор доступа: public

Метод: handler_5

Функционал: Функция приёмника

Параметры: string command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода handler_5 класса FifLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal to " + getAbsolutePath(this) + " Text: " + command	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: signal_0

Функционал: Функция сигнала

Параметры: string& command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода signal_0 класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal from " + getAbsolutePath(this)	2	
2		K command прибавить " (class: 1)"	∅	

Класс объекта: Tree

Модификатор доступа: public

Метод: handler_0

Функционал: Функция приёмника

Параметры: string command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода handler_0 класса Tree

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal to " +	∅	

		getAbsolutePath(this) + " Text: " + command		

Класс объекта: Leaf

Модификатор доступа: public

Метод: chooseSignal

Функционал: Выбор правильной функции сигнала класса

Параметры: int num

Возвращаемое значение: SIGNAL

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода chooseSignal класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1	num == 0	Вернуть SIGNAL_D(Tree::signal_0)	∅	
	num == 1	Вернуть SIGNAL_D(FLevel::signal_1)	∅	
	num == 2	Вернуть SIGNAL_D(SLevel::signal_2)	∅	
	num == 3	Вернуть SIGNAL_D(TLevel::signal_3)	∅	
	num == 4	Вернуть SIGNAL_D(FoLevel::signal_4)	∅	
	num == 5	Вернуть SIGNAL_D(FifLevel::signal_5)	∅	

Класс объекта: Leaf

Модификатор доступа: public

Метод: chooseHandler

Функционал: Выбор правильной функции приёмника класса

Параметры: int num

Возвращаемое значение: HANDLER

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода chooseHandler класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1	num == 0	Вернуть HANDLER_D(Tree::handler_0)	∅	
	num == 1	Вернуть HANDLER_D(FLevel::handler_1)	∅	
	num == 2	Вернуть HANDLER_D(SLevel::handler_2)	∅	
	num == 3	Вернуть HANDLER_D(TLevel::handler_3)	∅	
	num == 4	Вернуть HANDLER_D(FoLevel::handler_4)	∅	
	num == 5	Вернуть HANDLER_D(FifLevel::handler_5)	∅	

Класс объекта: Leaf

Модификатор доступа: public

Метод: setAllPrepaired

Функционал: Устанавливает параметр и всех его детей готовыми

Параметры: Leaf* leaf

Возвращаемое значение: void

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода setAllPrepaired класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление vector<Leaf*> vec = leaf->getChilds()	2	
2		Объявление vector<Leaf*>::iterator it = vec.begin()	3	
3	!vec.empty()	leaf->state = 1	4	
		leaf->state = 1	∅	
4	it != vec.end()	setAllPrepaired(*it)	5	
			∅	
5		it++	4	

Класс объекта: Leaf

Модификатор доступа: public

Метод: setAllNotPrepaired

Функционал: Устанавливает параметр и всех его детей готовыми

Параметры: Leaf* leaf

Возвращаемое значение: void

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода setAllNotPrepaired класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление vector<Leaf*> vec = leaf->getChilds()	2	
2		Объявление	3	

		vector<Leaf*>::iterator it = vec.begin()		
3	!vec.empty()	leaf->state = 0	4	
		leaf->state = 0	∅	
4	it != vec.end()	setAllNotPrepaired(*it)	5	
			∅	
5		it++	4	

Класс объекта: Leaf

Модификатор доступа: public

Метод: readCommands

Функционал: Считывает комманды

Параметры: map<string, Leaf*> treeMap

Возвращаемое значение: void

Алгоритм метода представлен в таблице 12.

Таблица 12. Алгоритм метода readCommands класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление string full_command, command, first_argument, second_argument	2	
2		Объявление map<string, Leaf*>::iterator it_map, it_map2	3	
3		Объявление SIGNAL ptr2Signal	4	
4		Объявление HANDLER ptr2Handler	5	
5		Объявление int num	6	

6		getline(cin, full_command)	7	
7		command = full_command.substr(0, full_command.find_first_of(' '))	8	
8	command != "END"	first_argument = full_command.substr (0, full_command.find_first_of(' '))	9	
			∅	
9	full_command != ""	second_argument = full_command.substr(full_command.find_first_of(' ') + 1)	10	
			10	
10	first_argument.size() != 1	it_map = treeMap.find(first_argument.substr (first_argument.find_last_of('/') + 1)	11	
		it_map = treeMap.find(this->name)	11	
11	second_argument.size() != 1	it_map2 = treeMap.find(second_argument.substr (second_argument.find_last_of('/') + 1))	12	
		it_map2 = treeMap.find(this->name)	12	
12	command == "EMIT"		13	
			15	
13	it_map->second->state != 0	ptr2Signal = chooseSignal(it_map->second->cls)	14	
			25	
14		it_map->second-	25	

		>emit_signal(ptr2Signal, second_argument)		
15	command == "SET_CONNECT"		16	
			19	
16	it_map->second->state != 0	ptr2Signal = chooseSignal(it_map->second->cls)	17	
			25	
17		ptr2Handler = chooseHandler(it_map2->second->cls)	18	
18		it_map->second->set_connect(ptr2Signal, it_map2->second, ptr2Handler)	25	
19	command == "DELETE_CONNECT"		20	
			23	
20	it_map->second->state != 0	ptr2Signal = chooseSignal(it_map->second->cls)	21	
			25	
21		ptr2Handler = chooseHandler(it_map2->second->cls)	22	
22		it_map->second->set_connect(ptr2Signal, it_map2->second, ptr2Handler)	25	
23	command == "SET_CONDITION"	num = stoi(second_argument)	24	
			25	
24	num != 0	it_map->second->state = num	25	
		setAllNotPrepaired(it_map-	25	

		>second)		
25		getline(cin, full_command)	26	
26		command = full_command.substr(0, full_command.find_first_of(' '))	27	
27		full_command = full_command.substr(full_command.find_first_of(' ') + 1)	8	

Класс объекта: Leaf

Модификатор доступа: public

Метод: getAbsolutePath

Функционал: Возвращает полный адрес объекта

Параметры: Leaf* leaf

Возвращаемое значение: string

Алгоритм метода представлен в таблице 13.

Таблица 13. Алгоритм метода getAbsolutePath класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление string absPath = object->getName()	2	
2	object->getParent()->getParent() != nullptr	object = object->getParent()	3	
		Вернуть '/' + absPath	∅	
3		absPath = object->getName() + '/' +	2	

		absPath		

Класс объекта: Leaf

Модификатор доступа: public

Метод: set_connect

Функционал: Устанавливает связь первого объекта со вторым объектам

Параметры: SIGNAL signal, Leaf* target, HANDLER handler

Возвращаемое значение: void

Алгоритм метода представлен в таблице 14.

Таблица 14. Алгоритм метода set_connect класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление o_sh* p_value	2	
2		Объявление unsigned int i = 0	3	
3	i < connects.size()		4	
		p_value = new o_sh()	5	
4	connects[i]->signal == signal && connects[i]->ptr == target && connects[i]->handler == handler		∅	
		К i прибавить 1	3	
5		p_value->signal = signal	6	
6		p_value->ptr = target	7	
7		p_value->handler = handler	8	
8		connects.push_back(p_value)	∅	

Класс объекта: Leaf

Модификатор доступа: public

Метод: delete_connect

Функционал: Удаляет связь первого объекта со вторым объектами

Параметры: SIGNAL signal, Leaf* target, HANDLER handler

Возвращаемое значение: void

Алгоритм метода представлен в таблице 15.

Таблица 15. Алгоритм метода delete_connect класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление unsigned int i = 0	2	
2	i < connects.size()		3	
			∅	
3	connects[i]->ptr = target	connects.erase(connects.begin() + i)	∅	
		К i прибавить 1	2	

Класс объекта: Leaf

Модификатор доступа: public

Метод: emit_signal

Функционал: Отправляет сообщение другим объектам

Параметры: SIGNAL signal, string& command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 16.

Таблица 16. Алгоритм метода emit_signal класса Leaf

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление HANDLER handler	2	
2	connects.empty()		∅	
			3	
3		(this->*signal)(command)	4	
4		Объявление unsigned int i = 0	5	
5	i < connects.size()		6	
			∅	
6	connects[i]->signal == signal	handler = connects[i]->handler	7	
			8	
7		(connects[i]->ptr->*handler)(command)	8	
8		К i прибавить 1	5	

Класс объекта: SLevel

Модификатор доступа: public

Метод: signal_2

Функционал: Функция сигнала

Параметры: string& command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 17.

Таблица 17. Алгоритм метода signal_2 класса SLevel

№	Предикат	Действия	№ перехода	Комментарий

1		Вывод "Signal from " + getAbsolutePath(this)	2	
2		K command прибавить " (class: 3)"	∅	

Класс объекта: TLevel

Модификатор доступа: public

Метод: signal_3

Функционал: Функция сигнала

Параметры: string& command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 18.

Таблица 18. Алгоритм метода signal_3 класса TLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal from " + getAbsolutePath(this)	2	
2		K command прибавить " (class: 4)"	∅	

Класс объекта: FoLevel

Модификатор доступа: public

Метод: signal_4

Функционал: Функция сигнала

Параметры: string& command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 19.

Таблица 19. Алгоритм метода signal_4 класса FoLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal from " + getAbsolutePath(this)	2	
2		K command прибавить " (class: 5)"	∅	

Класс объекта: FifLevel

Модификатор доступа: public

Метод: signal_5

Функционал: Функция сигнала

Параметры: string& command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 20.

Таблица 20. Алгоритм метода signal_5 класса FifLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal from " + getAbsolutePath(this)	2	
2		K command прибавить " (class: 5)"	∅	

Класс объекта: SLevel

Модификатор доступа: public

Метод: handler_2

Функционал: Функция приёмника

Параметры: string command

Возвращаемое значение: void

Алгоритм метода представлен в таблице 21.

Таблица 21. Алгоритм метода handler_2 класса SLevel

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "Signal to " + getAbsolutePath(this) + " Text: " + command	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

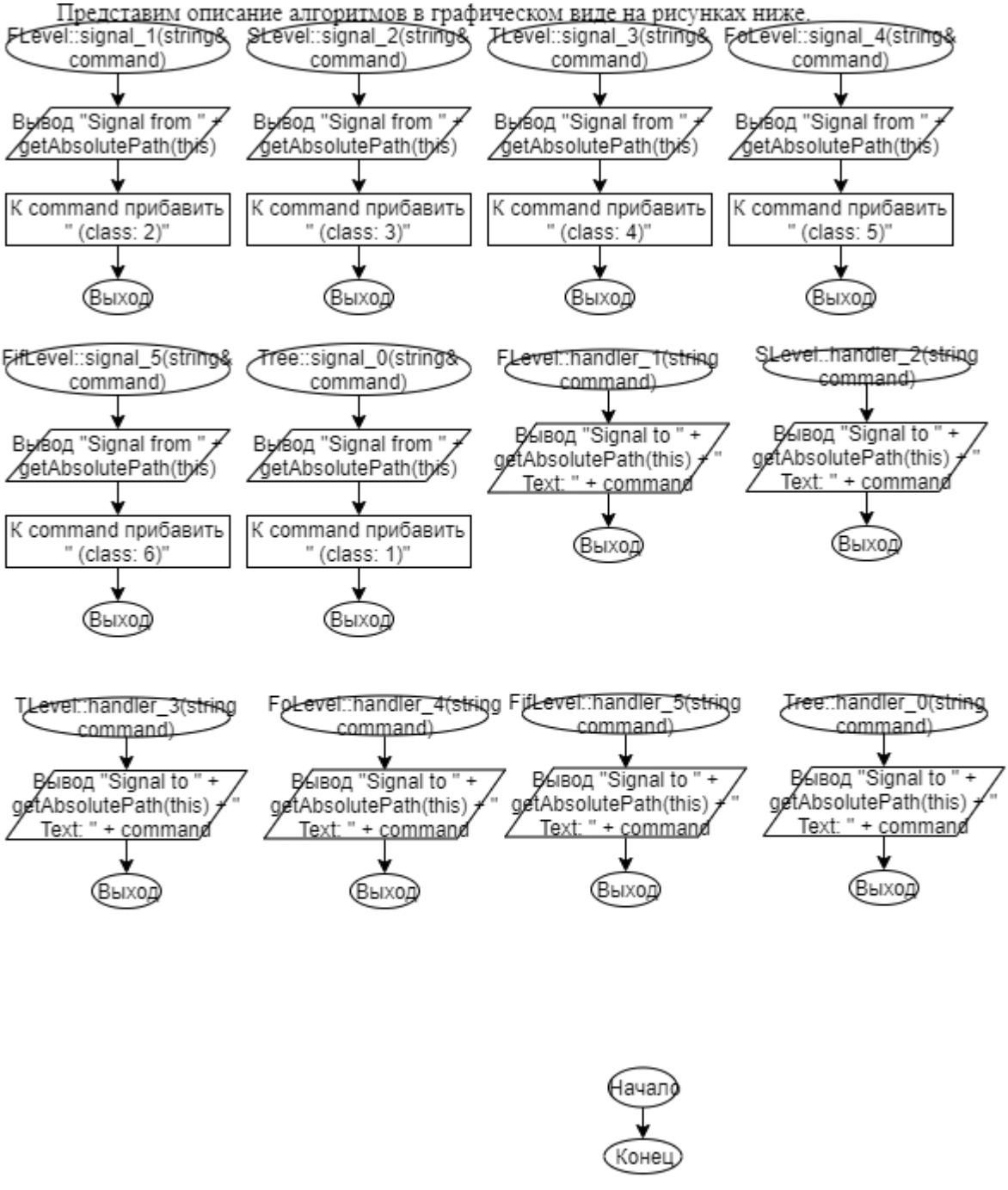

Рис. 1. Блок-схема алгоритма.

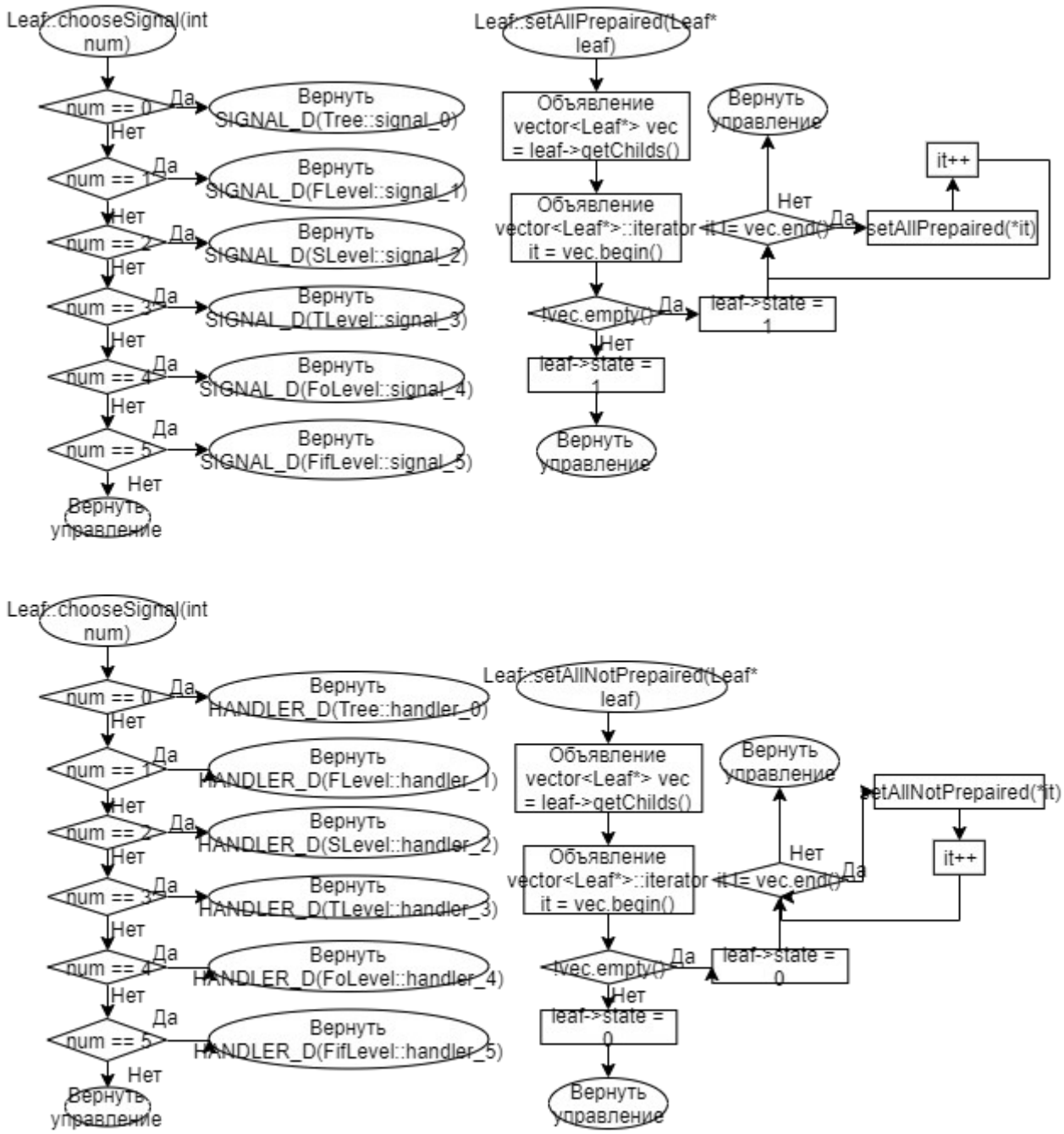

Рис. 2. Блок-схема алгоритма.

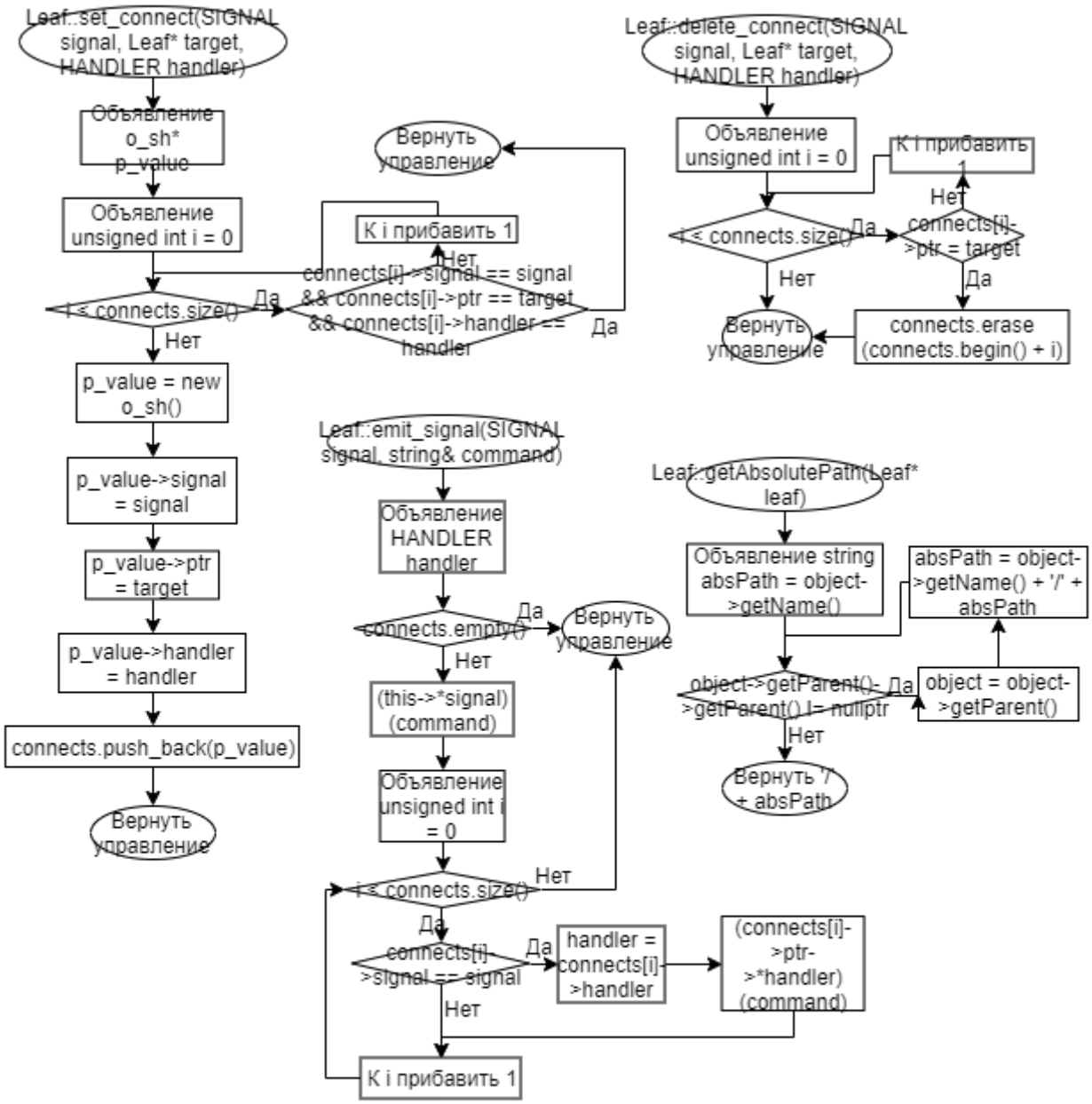

Рис. 3. Блок-схема алгоритма.

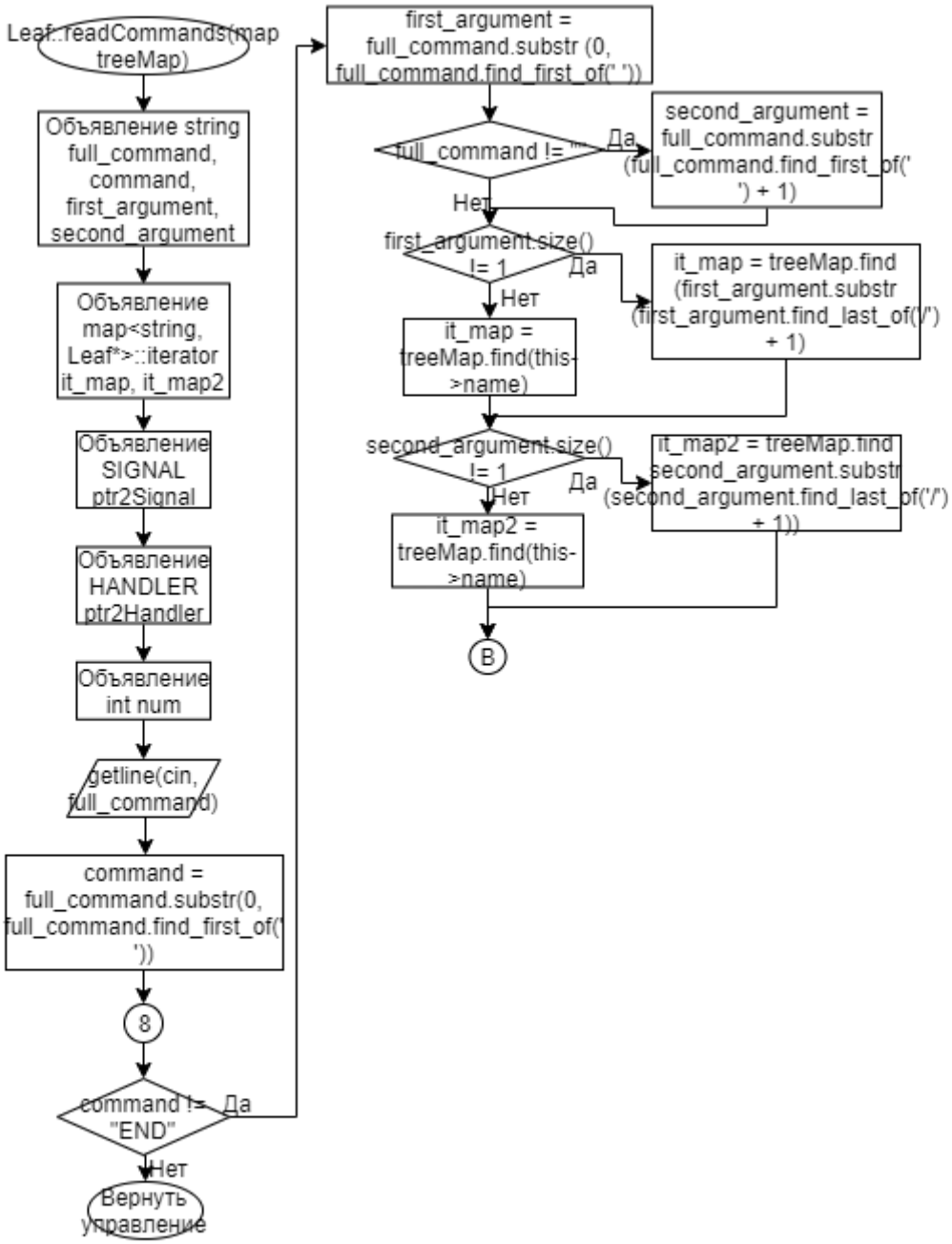

Рис. 4. Блок-схема алгоритма.

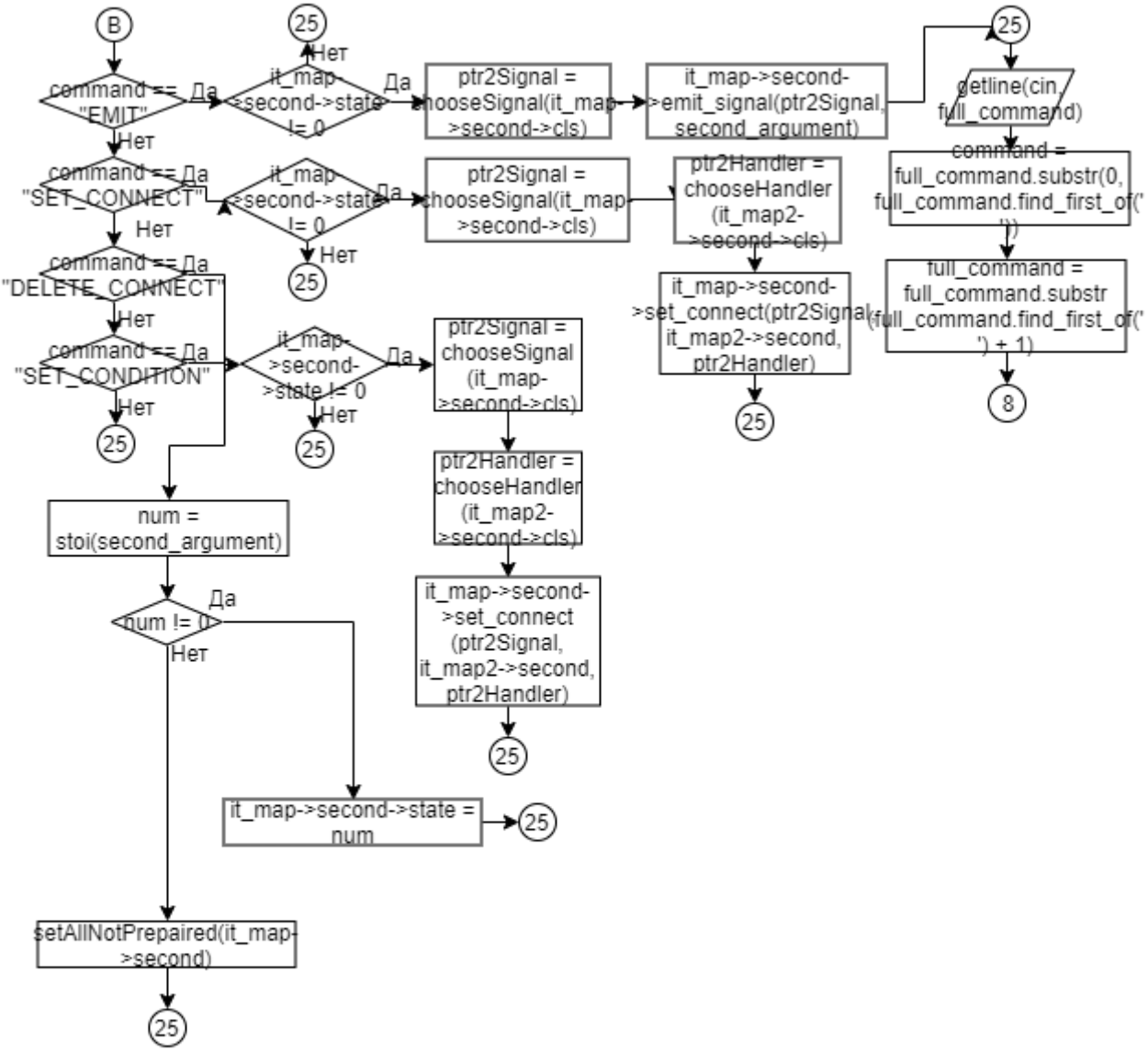

Рис. 5. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл FifLevel.cpp

```
#include "FifLevel.h"

void FifLevel::signal_5(string& command) {
	cout << endl << "Signal from " << getAbsolutePath(this);
	command += " (class: 6)";
}

void FifLevel::handler_5(string command) {
	cout << endl << "Signal to " << getAbsolutePath(this) << " Text:  " <<
command;
}
```

Файл FifLevel.h

```
#ifndef FIFLEVEL
#define FIFLEVEL

#include "Leaf.h"

class FifLevel : public Leaf {
public:
	FifLevel(Leaf* parent, string name) : Leaf(parent, name) {}
	void signal_5(string&);
	void handler_5(string);
};

#endif
```

Файл FLevel.cpp

```
#include "FLevel.h"

void FLevel::signal_1(string& command) {
	cout << endl << "Signal from " << getAbsolutePath(this);
	command += " (class: 2)";
}
```

```
void FLevel::handler_1(string command) {
	cout << endl << "Signal to " << getAbsolutePath(this) << " Text:  " <<
command;
}
```

Файл FLevel.h

```
#ifndef FLEVEL
#define FLEVEL

#include "Leaf.h"

class FLevel : public Leaf {
public:
	FLevel(Leaf* parent, string name) : Leaf(parent, name) {}
	void signal_1(string&);
	void handler_1(string);
};

#endif
```

Файл FoLevel.cpp

```
#include "FoLevel.h"

void FoLevel::signal_4(string& command) {
	cout << endl << "Signal from " << getAbsolutePath(this);
	command += " (class: 5)";
}

void FoLevel::handler_4(string command) {
	cout << endl << "Signal to " << getAbsolutePath(this) << " Text:  " <<
command;
}
```

Файл FoLevel.h

```
#ifndef FOLEVEL
#define FOLEVEL

#include "Leaf.h"

class FoLevel : public Leaf {
public:
	FoLevel(Leaf* parent, string name) : Leaf(parent, name) {}
	void signal_4(string&);
	void handler_4(string);
```

```
};

#endif
```

Файл Leaf.cpp

```
#include "Tree.h"
#include "Leaf.h"
#include "FLevel.h"
#include "SLevel.h"
#include "TLevel.h"
#include "FoLevel.h"
#include "FifLevel.h"

Leaf::Leaf(Leaf* parent, string name) : parent(parent), name(name) {};

Leaf::Leaf(Leaf* parent) : parent(parent) {
        string name;
        cin >> name;
        this->name = name;
}

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

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

string Leaf::getName() {
        return this->name;
}

Leaf* Leaf::getParent() {
        return this->parent;
}

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

void Leaf::addChild(Leaf* child) {
        this->childrens.push_back(child);
}

vector<Leaf*> Leaf::getChilds() {
        return this->childrens;
}

Leaf* Leaf::findChild(map<string, Leaf*> treeMap, Leaf* object) {
        map < string, Leaf* >::iterator got;
        got = treeMap.find(object->name);
```

```
        return (got == treeMap.end()) ? got->second : nullptr;
}

void Leaf::printLeaf(Leaf* leaf) {
        static int level = 0;
        vector<Leaf*>::iterator it;
        vector<Leaf*> vec = leaf->getChilds();
        if (!vec.empty()) {
                cout << endl;
                for (int i = 0; i < level; i++) cout << "    ";
                cout << leaf->getName();
                level++;
                for (it = vec.begin(); it != vec.end(); it++) {
                        printLeaf(*it);
                }
                level--;
        } else {
                cout << endl;
                for (int i = 0; i < level; i++) cout << "    ";
                cout << leaf->name;
        }
}

void Leaf::printLeafPrepaired(Leaf* leaf) {
        static int level = 0;
        vector<Leaf*>::iterator it;
        vector<Leaf*> vec = leaf->getChilds();
        if (!vec.empty()) {
                cout << endl;
                for (int i = 0; i < level; i++) cout << "    ";
                cout << leaf->getName();
                if (leaf->state != 0) {
                        cout << " is ready";
                } else {
                        cout << " is not ready";
                }
                level++;
                for (it = vec.begin(); it != vec.end(); it++) {
                        printLeafPrepaired(*it);
                }
                level--;
        } else {
                cout << endl;
                for (int i = 0; i < level; i++) cout << "    ";
                cout << leaf->name;
                if (leaf->state != 0) {
                        cout << " is ready";
                } else {
                        cout << " is not ready";
                }
        }
}

void Leaf::setPrepaired(map<string, Leaf*> treeMap) {
        string str;
        Leaf*ptr;
        map<string, Leaf*>::iterator got;
        int num;
        bool flag = false;
        while (cin >> str >> num) {
```

```
                got = treeMap.find(str);
                ptr = got->second;
                if (got != treeMap.end()) {
                        if (ptr->parent == nullptr) {
                                flag = true;
                        } else {
                                while (true) {
                                        ptr = ptr->parent;
                                        if (ptr == nullptr) break;
                                        if (ptr->state == 0) {
                                                flag = false;
                                                break;
                                        } else {
                                                flag = true;
                                        }
                                }
                        }
                        if (flag) {
                                if (num != 0) {
                                        got->second->state = num;
                                } else {
                                        remRead(got->second);
                                }
                        }
                }
        }
}

void Leaf::readOperations(map<string, Leaf*> treeMap) {
        string command, adress, objectName, tmp, tmp2;
        map<string, Leaf*>::iterator ptr;
        bool notFound;
        size_t f;
        int out;
        Leaf* currentObject = this;
        Leaf* objPtr;

        cin >> command;
        while(command != "END") {
                objPtr = currentObject;
                notFound = false;
                cin >> adress;
                objectName = adress.substr(adress.find_last_of('/') + 1);
                tmp = adress;

                if (adress == ".") {
                        objPtr = currentObject;
                        tmp = currentObject->name;
                        objectName = currentObject->name;
                } else if (adress.at(0) == '/' && adress.size() == 1) {
                        objPtr = this;
                        tmp = objPtr->name;
                        objectName = this->name;
                } else if (adress.at(0) == '/' && adress.at(1) != '/') {
                        objPtr = this;
                        tmp = tmp.substr(1);
                } else if (adress.substr(0, 2) == "//") {
```

```
                        ptr = treeMap.find(objectName);
                        if (command == "SET") {
                                if (ptr != treeMap.end()) {
                                        currentObject = ptr->second;
                                        cout << endl << "Object is set: " <<
objectName;
                                } else {
                                        cout << endl << "Object is not found:
" << currentObject->name << ' ' << adress;
                                }
                        } else if (command == "FIND") {
                                if (ptr != treeMap.end()) {
                                        cout << endl << adress << "     Object
name: " << objectName;
                                } else {
                                        cout << endl << adress << "     Object
is not found";
                                }
                        }
                        cin >> command;
                        continue;
                }

                tmp2 = tmp;
                f = tmp.find_first_of('/');
                while (f != string::npos) {
                        tmp2 = tmp.substr(0, f);
                        out = vecFind(objPtr, tmp2);
                        if (out == -1) {
                                notFound = true;
                                break;
                        }
                        objPtr = objPtr->getChilds().at(out);
                        tmp = tmp.substr(f + 1);
                        f = tmp.find_first_of('/');
                }
                out = vecFind(objPtr, tmp);
                if (out == -1) {
                        notFound = true;
                } else if (out == -2) {
                        notFound = false;
                } else {
                        objPtr = objPtr->getChilds().at(out);
                }

                if (command == "SET") {
                        if (notFound) {
                                cout << endl << "Object is not found: " <<
currentObject->name << ' ' << adress;
                        } else {
                                currentObject = objPtr;
                                cout << endl << "Object is set: " <<
objectName;
                        }
                } else if (command == "FIND") {
                        if (notFound) {
                                cout << endl << adress << "     Object is not
found";
                        } else {
                                cout << endl << adress << "     Object name: "
```

```
<< objectName;
                        }
                }
                cin >> command;
        }
}

int Leaf::vecFind(Leaf* objPtr, string name) {
        vector<Leaf*> vec = objPtr->getChilds();
        vector<Leaf*>::iterator it;
        int counter = 0;
        if (!vec.empty()) {
                for (it = vec.begin(); it != vec.end(); it++, counter++) {
                        if ((*it)->getName() == name) return counter;
                }
        }
        if (objPtr->name == name) return -2;
        return -1;
}

void Leaf::remRead(Leaf* ptr) {
        vector<Leaf*> vec = ptr->getChilds();
        vector<Leaf*>::iterator it;
        ptr->setState(0);
        if (!vec.empty()) for (it = vec.begin(); it != vec.end(); it++)
remRead(*it);
}

SIGNAL Leaf::chooseSignal(int num) {
        switch(num) {
                case 0:
                        return SIGNAL_D(Tree::signal_0);
                case 1:
                        return SIGNAL_D(FLevel::signal_1);
                case 2:
                        return SIGNAL_D(SLevel::signal_2);
                case 3:
                        return SIGNAL_D(TLevel::signal_3);
                case 4:
                        return SIGNAL_D(FoLevel::signal_4);
                case 5:
                        return SIGNAL_D(FifLevel::signal_5);
        }
        return nullptr;
}

HANDLER Leaf::chooseHandler(int num) {
        switch(num) {
                case 0:
                        return HANDLER_D(Tree::handler_0);
                case 1:
                        return HANDLER_D(FLevel::handler_1);
                case 2:
                        return HANDLER_D(SLevel::handler_2);
                case 3:
                        return HANDLER_D(TLevel::handler_3);
                case 4:
```

```
                        return HANDLER_D(FoLevel::handler_4);
                case 5:
                        return HANDLER_D(FifLevel::handler_5);
        }
        return nullptr;
}

void Leaf::setAllPrepaired(Leaf* leaf) {
        vector<Leaf*> vec = leaf->getChilds();
        vector<Leaf*>::iterator it;
        if (!vec.empty()) {
                leaf->state = 1;
                for (it = vec.begin(); it != vec.end(); it++) {
                        setAllPrepaired(*it);
                }
        } else {
                leaf->state = 1;
        }
}

void Leaf::setAllNotPrepaired(Leaf* leaf) {
        vector<Leaf*> vec = leaf->getChilds();
        vector<Leaf*>::iterator it;
        if (!vec.empty()) {
                leaf->state = 0;
                for (it = vec.begin(); it != vec.end(); it++) {
                        setAllNotPrepaired(*it);
                }
        } else {
                leaf->state = 0;
        }
}

void Leaf::readCommands(map<string, Leaf*> treeMap) {
        string full_command, command, first_argument, second_argument;
        map<string, Leaf*>::iterator it_map, it_map2;
        SIGNAL ptr2Signal;
        HANDLER ptr2Handler;
        int num;

        getline(cin, full_command);
        command = full_command.substr(0, full_command.find_first_of(' '));
        full_command = full_command.substr(full_command.find_first_of(' ') +
1);
        while (command != "END") {
                if (full_command != "") {
                        first_argument = full_command.substr(0,
full_command.find_first_of(' '));
                        second_argument =
full_command.substr(full_command.find_first_of(' ') + 1);
                        if (first_argument.substr(0, 2) == "//" &&
first_argument.size() >= 2) {
                                it_map =
treeMap.find(first_argument.substr(first_argument.find_last_of('/') + 1));
                        } else if (first_argument.size() != 1) {
                                if (isCorrect(first_argument)) {
                                        it_map =
treeMap.find(first_argument.substr(first_argument.find_last_of('/') + 1));
                                } else {
                                        it_map = treeMap.end();
```

```
                        }
                    } else {
                        it_map = treeMap.find(this->name);
                    }
                    if (second_argument.substr(0, 2) == "//" &&
second_argument.size() >= 2) {
                        it_map2 =
treeMap.find(second_argument.substr(second_argument.find_last_of('/') + 1));
                    } else if (second_argument.size() != 1) {
                        if (isCorrect(second_argument)
                        && second_argument.find_last_of(' ') ==
second_argument.find_first_of(' ')) {
                            it_map2 =
treeMap.find(second_argument.substr(second_argument.find_last_of('/') + 1));
                        } else {
                            it_map2 = treeMap.end();
                        }
                    } else {
                        it_map2 = treeMap.find(this->name);
                    }
                    if (it_map == treeMap.end()) {
                        cout << endl << "Object " << first_argument <<
" not found";
                    } else if (it_map2 == treeMap.end()
                        && second_argument.at(0) == '/') {
                        cout << endl << "Handler object " <<
second_argument << " not found";
                    } else if (command == "EMIT") {
                        if (it_map->second->state != 0) {
                            ptr2Signal = chooseSignal(it_map-
>second->cls);
                            it_map->second-
>emit_signal(ptr2Signal, second_argument);
                        }
                    } else if (command == "SET_CONNECT") {
                        if (it_map->second->state != 0 && it_map2-
>second->state != 0) {
                            ptr2Signal = chooseSignal(it_map-
>second->cls);
                            ptr2Handler = chooseHandler(it_map2-
>second->cls);
                            it_map->second-
>set_connect(ptr2Signal, it_map2->second, ptr2Handler);
                        }
                    } else if (command == "DELETE_CONNECT") {
                        if (it_map->second->state != 0 && it_map2-
>second->state != 0) {
                            ptr2Signal = chooseSignal(it_map-
>second->cls);
                            ptr2Handler = chooseHandler(it_map2-
>second->cls);
                            it_map->second-
>delete_connect(ptr2Signal, it_map2->second, ptr2Handler);
                        }
                    } else if (command == "SET_CONDITION") {
                        num = stoi(second_argument);
                        if (num != 0) {
```

```
                                        it_map->second->state = num;
                                } else {
                                        setAllNotPrepaired(it_map->second);
                                }
                        }
                }
                getline(cin, full_command);
                command = full_command.substr(0, full_command.find_first_of('
'));
                full_command =
full_command.substr(full_command.find_first_of(' ') + 1);
        }
}

string Leaf::getAbsolutePath(Leaf* object) {
        string absPath = object->getName();
        while(object->getParent()->getParent() != nullptr) {
                object = object->getParent();
                absPath = object->getName() + '/' + absPath;
        }
        return('/' + absPath);
}

void Leaf::set_connect(SIGNAL signal, Leaf* target, HANDLER handler) {
        o_sh* p_value;

        for (unsigned int i = 0; i < connects.size(); i++) {
                if (connects[i]->signal == signal &&
                        connects[i]->ptr == target &&
                        connects[i]->handler == handler) {
                        return;
                }
        }
        p_value = new o_sh();
        p_value->signal = signal;
        p_value->ptr = target;
        p_value->handler = handler;
        connects.push_back(p_value);
}

void Leaf::delete_connect(SIGNAL signal, Leaf* target, HANDLER handler) {
        for (unsigned int i = 0; i < connects.size(); i++) {
                if (connects[i]->ptr == target) {
                        connects.erase(connects.begin() + i);
                        return;
                }
        }
}

void Leaf::emit_signal(SIGNAL signal, string& command) {
        HANDLER handler;

        (this->*signal)(command);

        if (connects.empty()) return;

        for (unsigned int i = 0; i < connects.size(); i++) {
                if (connects[i]->signal == signal) {
                        handler = connects[i]->handler;
                        (connects[i]->ptr->*handler)(command);
```

```
                }
        }
}

bool Leaf::isCorrect(string coordinate) {
        string tmp;
        size_t f;
        int out;
        Leaf* objPtr = this;

        coordinate = coordinate.substr(1);
        tmp = coordinate;
        f = tmp.find_first_of('/');
        while (f != string::npos) {
                tmp = coordinate.substr(0, f);
                out = vecFind(objPtr, tmp);
                if (out == -1) return false;
                objPtr = objPtr->getChilds().at(out);
                coordinate = coordinate.substr(f + 1);
                f = coordinate.find_first_of('/');
        }
        return true;
}
```

Файл Leaf.h

```
#ifndef LEAF
#define LEAF

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

using namespace std;

class Leaf;

typedef void(Leaf::* SIGNAL) (string&);
typedef void(Leaf::* HANDLER) (string);

#define SIGNAL_D(signal) (SIGNAL) (&signal)
#define HANDLER_D(handler) (HANDLER) (&handler)

class Leaf {
        string name;
        Leaf* parent;
        vector<Leaf*> childrens;
        int state = 0;

        struct o_sh {
                SIGNAL signal;
                Leaf* ptr;
```

```
                HANDLER handler;
        };
        vector <o_sh*> connects;

        int vecFind(Leaf* objPtr, string name);
        void remRead(Leaf* ptr);
public:
        int cls = -1;

        Leaf(Leaf* parent, string name);
        Leaf(Leaf* parent);
        void setName(string name);
        void setParent(Leaf* parent);
        string getName();
        Leaf* getParent();
        void setState(int state);
        void addChild(Leaf* child);
        vector<Leaf*> getChilds();
        Leaf* findChild(map<string, Leaf*> treeMap, Leaf* object);
        void printLeaf(Leaf* leaf);
        void printLeafPrepaired(Leaf* leaf);
        void setPrepaired(map<string, Leaf*> treeMap);
        void readOperations(map<string, Leaf*> treeMap);

        string getAbsolutePath(Leaf* object);
        void set_connect(SIGNAL signal, Leaf* target, HANDLER handler);
        void delete_connect(SIGNAL signal, Leaf* target, HANDLER handler);
        void emit_signal(SIGNAL signal, string& command);
        void setAllPrepaired(Leaf* leaf);
        void setAllNotPrepaired(Leaf* leaf);
        void readCommands(map<string, Leaf*> treeMap);
        SIGNAL chooseSignal(int num);
        HANDLER chooseHandler(int num);
        bool isCorrect(string coordinate);
};

#endif
```

Файл main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "Leaf.h"
#include "Tree.h"

int main()
{
        Tree tree(nullptr);
        tree.buildTree();
        return tree.execApp();
}
```

Файл SLevel.cpp

```
#include "SLevel.h"

void SLevel::signal_2(string& command) {
        cout << endl << "Signal from " << getAbsolutePath(this);
        command += " (class: 3)";
}

void SLevel::handler_2(string command) {
        cout << endl << "Signal to " << getAbsolutePath(this) << " Text:  " <<
command;
}
```

Файл SLevel.h

```
#ifndef SLEVEL
#define SLEVEL

#include "Leaf.h"

class SLevel : public Leaf {
public:
        SLevel(Leaf* parent, string name) : Leaf(parent, name) {}
        void signal_2(string&);
        void handler_2(string);
};

#endif
```

Файл TLevel.cpp

```
#include "TLevel.h"

void TLevel::signal_3(string& command) {
        cout << endl << "Signal from " << getAbsolutePath(this);
        command += " (class: 4)";
}

void TLevel::handler_3(string command) {
        cout << endl << "Signal to " << getAbsolutePath(this) << " Text:  " <<
command;
}
```

Файл TLevel.h

```
#ifndef TLEVEL
#define TLEVEL

#include "Leaf.h"

class TLevel : public Leaf {
public:
        TLevel(Leaf* parent, string name) : Leaf(parent, name) {}
        void signal_3(string&);
        void handler_3(string);
};

#endif
```

Файл Tree.cpp

```
#include "Tree.h"
#include "Leaf.h"
#include "FLevel.h"
#include "SLevel.h"
#include "TLevel.h"
#include "FoLevel.h"
#include "FifLevel.h"

using namespace std;

void Tree::buildTree() {
        treeMap.insert(pair<string, Leaf*>(this->getName(), this));
        map < string, Leaf* >::iterator got, it_head, it_tail;
        Leaf* leafPtr;
        string head, tail, tmp;
        int numClass;

        this->cls = 0;

        cin >> head;
        while (head != "endtree") {
                tmp = head;
                if (head == "/") {
                        head = this->getName();
                } else if (head.size() > 1 && head.at(0) == '/') {
                        head = head.substr(head.find_last_of('/') + 1);
                }
                cin >> tail >> numClass;
                got = treeMap.find(head);
                if (got != treeMap.end()) {
                        switch(numClass) {
                                case 2:
                                        leafPtr = new FLevel(got->second,
tail);
```

```
                                        leafPtr->cls = 1;
                                        break;

                                case 3:
                                        leafPtr = new SLevel(got->second,
tail);
                                        leafPtr->cls = 2;
                                        break;

                                case 4:
                                        leafPtr = new TLevel(got->second,
tail);
                                        leafPtr->cls = 3;
                                        break;

                                case 5:
                                        leafPtr = new FoLevel(got->second,
tail);
                                        leafPtr->cls = 4;
                                        break;

                                case 6:
                                        leafPtr = new FifLevel(got->second,
tail);
                                        leafPtr->cls = 5;
                                        break;
                        }
                        treeMap.insert(pair<string, Leaf*>(tail, leafPtr));
                        got->second->addChild(leafPtr);
                } else {
                        cout << "Object tree";
                        printLeaf(this);
                        cout << endl << "The head object " << tmp << " is not
found";
                        this->error = true;
                }
                cin >> head;
        }

        SIGNAL ptr2Signal;
        HANDLER ptr2Handler;

        cin >> head;
        while (head != "end_of_connections") {
                cin >> tail;


                if (head.substr(0, 2) == "//" && head.size() >= 2) {
                                it_head =
treeMap.find(head.substr(head.find_last_of('/') + 1));
                        } else if (head.size() != 1) {
                                if (isCorrect(head)) {
                                        it_head =
treeMap.find(head.substr(head.find_last_of('/') + 1));
                                } else {
                                        it_head = treeMap.end();
                                }
```

```
                        } else {
                                it_head = treeMap.find(this->getName());
                        }
                        if (tail.substr(0, 2) == "//" && tail.size() >= 2) {
                                it_tail =
treeMap.find(tail.substr(tail.find_last_of('/') + 1));
                        } else if (tail.size() != 1) {
                                if (isCorrect(tail)
                                && tail.find_last_of(' ') ==
tail.find_first_of(' ')) {
                                        it_tail =
treeMap.find(tail.substr(tail.find_last_of('/') + 1));
                                } else {
                                        it_tail = treeMap.end();
                                }
                        } else {
                                it_tail = treeMap.find(this->getName());
                        }


                /*if (head == "/") {
                        head = this->getName();
                } else if (head.size() > 1 && head.at(0) == '/') {
                        head = head.substr(head.find_last_of('/') + 1);
                }
                it_head = treeMap.find(head);
                if (tail == "/") {
                        tail = this->getName();
                } else if (tail.size() > 1 && tail.at(0) == '/') {
                        tail = tail.substr(tail.find_last_of('/') + 1);
                }
                it_tail = treeMap.find(tail);*/
                if (it_head != treeMap.end() && it_tail != treeMap.end()) {
                        ptr2Signal = chooseSignal(it_head->second->cls);
                        ptr2Handler = chooseHandler(it_tail->second->cls);
                        it_head->second->set_connect(ptr2Signal, it_tail-
>second, ptr2Handler);
                } else if (it_head == treeMap.end()) {
                        cout << endl << "Object " << head << " not found";
                } else if (it_tail == treeMap.end()) {
                        cout << endl << "Handler object " << tail << " not
found";
                }
                cin >> head;
        }
}

bool Tree::execApp() {
        if (!this->error) {
                cout << "Object tree";
                printLeaf(this);
                setAllPrepaired(this);
                readCommands(this->treeMap);
        }
        return false;
}

void Tree::signal_0(string& command) {
        cout << endl << "Signal from " << getAbsolutePath(this);
        command += " (class: 1)";
```

```
}

void Tree::handler_0(string command) {
        cout << endl << "Signal to / Text:  " << command;
}
```

Файл Tree.h

```
#ifndef TREE
#define TREE
#include "Leaf.h"
#include <string>
#include <iostream>
#include <map>

class Tree: public Leaf {
private:
        map < string, Leaf* > treeMap;
        vector < string > commands;
        bool error = false;
public:
        void buildTree();
        bool execApp();
        void signal_0(string& command);
        void handler_0(string command);

        Tree(Leaf* parent) : Leaf(parent) {}
};

#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
appls_root / object_s1 3 / object_s2 2 /object_s2 object_s4 4 / object_s13 5 /object_s2 object_s6 6 /object_s1 object_s7 2 endtree /object_s2/object_s4 /object_s2/object_s6 /object_s2 /object_s1/object_s7 / /object_s2/object_s4 /object_s2/object_s4 / end_of_connections EMIT /object_s2/object_s4 Send message 1 DELETE_CONNECT /object_s2/object_s4 / EMIT /object_s2/object_s4 Send message 2 SET_CONDITION /object_s2/object_s4 0 EMIT /object_s2/object_s4 Send message 3 SET_CONNECT /object_s1 /object_s2/object_s6 EMIT /object_s1 Send message 4 END	Object tree appls_root object_s1 object_s7 object_s2 object_s4 object_s6 object_s13 Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4) Signal from /object_s2/object_s4 Signal to /object_s2/object_s6 Text: Send message 2 (class: 4) Signal from /object_s1 Signal to /object_s2/object_s6 Text: Send message 4 (class: 3)	

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (ИСТОЧНИКОВ)

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).