# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ .....................................................................................................................5](#_Toc136679442)

[1 Постановка задачи .......................................................................................................5](#_Toc136679443)

[2 Метод решения ...........................................................................................................12](#_Toc136679444)

[3 Описание алгоритмов .................................................................................................14](#_Toc136679445)

[4 Блок-схемы алгоритмов .............................................................................................23](#_Toc136679446)

[ЗАКЛЮЧЕНИЕ .............................................................................................................36](#_Toc136679447)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ....................................................37](#_Toc136679448)

[Приложение 1. Код программы ....................................................................................38](#_Toc136679449)

[Приложение 2. Тестирование .....................................................................................54](#_Toc136679450)

# ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

Механизм сигналов и обработчиков является одним из способов организации взаимодействия между объектами в программировании.Он позволяет установить связь между сигналами, которые генерируются объектом, и обработчиками, которые реагируют на эти сигналы. Вместе с передачей сигнала также передаются определенные данные, что делает механизм сигналов и обработчиков мощным инструментом для реализации сложных схем взаимодействия объектов.

В данной работе мы рассмотрим задачу организации взаимосвязи объектов посредством механизма сигналов и обработчиков. Мы опишем алгоритм и метод решения этой задачи, а также представим пример кода, демонстрирующего работу данного механизма. Основная цель работы - исследовать и объяснить принципы работы механизма сигналов и обработчиков, а также продемонстрировать его применение на практике.

В ходе работы мы описываем три основных метода, необходимых для организации взаимосвязи по механизму сигналов и обработчиков: метод установки связи, метод удаления связи и метод выдачи сигнала. Мы предлагаем использование указателей на методы сигнала и обработчика, что позволяет гибко устанавливать и разрывать связи между объектами.Также мы предлагаемиспользовать структуру для хранения установленных связей и вектор для управления ими.

# 1 Постановка задачи

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:

* установления связи между сигналом текущего объекта и обработчиком целевого объекта;
* удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
* 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:

* Если текущий объект отключен, то выход, иначе к пункту 2;
* Вызов метода сигнала с передачей строковой переменной по ссылке;
* Цикл по всем связям сигнал-обработчик текущего объекта:
  + Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то проверить готовность целевого объекта. Если целевой объект готов, то вызвать метод обработчика целевого объекта указанной в связи и передать в качестве аргумента строковую переменную по значению.
* Конец цикла.

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

В базовый класс добавить метод определения абсолютной 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. Если при построении дерева иерархии возникает ситуация дубляжа имен среди починенных у текущего головного объекта, то новый объект не создается.

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»

Моделировать работу системы, которая выполняет следующие команды с параметрами:

* EMIT «координата объекта» «текст» – выдает сигнал от заданного по координате объекта;
* SET\_CONNECT «координата объекта, выдающего сигнал» «координата целевого объекта» – устанавливает связь;
* DELETE\_CONNECT «координата объекта, выдающего сигнал» «координата целевого объекта» – удаляет связь;
* SET\_CONDITION «координата объекта» «значение состояния» – устанавливает состояние объекта;
* END – завершает функционирование системы (выполнение программы).

Реализовать алгоритм работы системы:

* в методе построения системы:
  + построение дерева иерархии объектов согласно вводу;
  + ввод и построение множества связей сигнал-обработчик для заданных пар объектов.
* в методе отработки системы:
  + привести все объекты в состоянии готовности;
  + цикл до признака завершения ввода:
    - ввод наименования объекта и текста сообщения;
    - вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной, содержащей текст сообщения.
  + 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.

## 1.1 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

EMIT /object\_s2/object\_s4 Send message 2

EMIT /object\_s2/object\_s4 Send message 3

EMIT /object\_s1 Send message 4

END

## 1.2 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 to / Text: Send message 2 (class: 4)

Signal from /object\_s2/object\_s4

Signal to /object\_s2/object\_s6 Text: Send message 3 (class: 4)

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

Signal from /object\_s1

# 2 Метод решения

Для решения задача было добавлено/изменено:

Класс cl\_base:

• Свойства/поля: нет изменений

• Методы:

• void set\_connect(TYPE\_SIGNAL p\_signal, cl\_base\* p\_object, TYPE\_HANDLER p\_ob\_handler) - создание связи между объектами, модификатор доступа public

• void delete\_connect(Указатель на метод Signal класса Base,Указатель на объект класса Base, Указатель на метод handler класса Base), удаление связи между объектами, модификатор доступа public

• void emit\_signal (TYPE\_SIGNAL p\_signal, string& s\_command)-Выдача сигнала , модификатор доступа publicstring get\_address() полученяи указателя на объект по его абсолютному адресу в дереев иерархии объектов, модификатор доступа public

Класс cl\_application:

• Свойства/поля: нет изменений

• Методы:

• void signal\_f (string s\_command)- метод Выводa сигнала, модификатор доступа public

• void handler\_f(string s\_command)- метод обработчика сигнала модификатор доступа public

Иерархия наследования отображена в таблице 1.

Таблица 1 – Иерархия наследования классов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Имя класса** | **Классы-**  **наследники** | **Модификатор доступа при**  **наследовании** | **Описание** | **Номер** |
| 1 | cl\_base |  |  | Базовый класс в иерархии наследования, содержит основной  функционал |  |
| cl\_application | рublic |  | 2 |
| cl\_2 | рublic |  | 3 |
| cl\_3 | рublic |  | 4 |
| cl\_4 | рublic |  | 5 |
| cl\_5 | рublic |  | 6 |
| cl\_6 | public |  | 7 |
| 2 | cl\_application |  |  | класс-приложение |  |
| 3 | cl\_2 |  |  | класс, наследуемый  от базового |  |
| 4 | cl\_3 |  |  | класс, наследуемый  от базового |  |
| 5 | cl\_4 |  |  | класс, наследуемый  от базового |  |
| 6 | cl\_5 |  |  | класс, наследуемый  от базового |  |
| 7 | cl\_6 |  |  | класс, наследуемый  от базового |  |

# 3 Описание алгоритмов

Согласно этапам разработки [3,4], после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

## Алгоритм функции main

Функционал: основная функция программы.

Параметры: отсутствуют.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 |  | создание объекта ob\_cl\_application класса cl\_application с помощью конструктора с параметром nullptr | 2 |
| 2 |  | вызов метода build\_tree\_objects() объекта ob\_cl\_application | 3 |
| 3 |  | вызов метода exec\_app() объекта ob\_cl\_application | ∅ |

## Алгоритм метода signal­\_f класса cl\_application

Функционал: метод сигнала метод сигнала методы классы cl\_1, cl\_2, cl\_3, cl\_4, cl\_5, cl\_6 аналогины класса cl\_application.

Параметры: string& msg.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода signal\_f класса cl\_application

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 |  | вывод символа переноса строки, строки "Signal from " и возвращенного значения метода get\_way() | 2 |
| 2 |  | прибавление строки " (class:1)" к переменной msg | ∅ |

## Алгоритм метода handler\_f класса cl\_application

Функционал: метод обработчика, методы классы cl\_1, cl\_2, cl\_3, cl\_4, cl\_5, cl\_6 аналогины класса cl\_application.

Параметры: string& msg.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода handler\_f класса cl\_application

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 |  | вывод символа переноса строки, строки "Signal to ", возвращенного значения метода get\_way(), строки "Text: " и параметра msg | ∅ |

## Алгоритм метода set\_connect класса cl\_base

Функционал: установление связи между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE\_SIGNAL p\_signal, cl\_base\*p\_object, TYPE\_HANDLER p\_handler.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода set\_connect класса cl\_base

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 |  | создание указателя p\_value на структуру o\_sh | 2 |
| 2 |  | инициализация целочисленной переменной i = 0 | 3 |
| 3 | i меньше размера вектора connects |  | 4 |
|  |  | 5 |
| 4 | p\_signal i-го элемента вектора connects равно параметру p\_signal и p\_cl\_base i-го элемента вектора connects равно параметру p\_object и p\_handler i-го элемента вектора connects равно параметру p\_handler |  | ∅ |
|  | увеличение i на 1 | 5 |
| 5 |  | создание объекта p\_value структуры o\_sh | 6 |
| 6 |  | поле p\_signal объекта p\_value равно параметру p\_signal | 7 |
| 7 |  | поле p\_cl\_base объекта p\_value равно параметру p\_object | 8 |
| 8 |  | поле p\_handler объекта p\_value равно параметру p\_handler | 9 |
| 9 |  | вызов метода push\_bask с параметром p\_value для вектора connects | ∅ |

## Алгоритм метода delete\_connect класса cl\_base

Функционал: удаление (разрыв) связи между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE\_SIGNAL p\_signal, cl\_base\*p\_object, TYPE\_HANDLER p\_handler.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода delete\_connect класса cl\_base

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 |  | инициализация целочисленной переменной i = 0 | 2 |
| 2 | i меньше размера вектора connects |  | 3 |
|  |  | ∅ |
| 3 | p\_signal i-го элемента вектора connects равно параметру p\_signal и p\_cl\_base i-го элемента вектора connects равно параметру p\_object и p\_handler i-го элемента вектора connects равно параметру p\_handler | вызов метода erase вектора connects с параметром возвращенное значение метода begin вектора connects + i | 4 |
|  |  | 4 |
| 4 |  | увеличение i на 1 | 2 |

## Алгоритм метода emit\_signal класса cl\_base

Функционал: выдача сигнала от текущего объекта с передачей строковой переменной.

Параметры: TYPE\_SIGNAL p\_signal, string& msg.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода emit\_signal класса cl\_base

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 | поле ready равно 0 |  | ∅ |
|  | вызов метода p\_signal с передачей переменной msg по ссылке | 2 |
| 2 | в векторе connects есть объекты |  | 3 |
|  |  | ∅ |
| 3 | поле p\_signal текущего объекта равно параметру p\_signal и поле p\_cl\_base имеет поле ready не равное 0 | инициализация указателя на базовый класс p\_target полем p\_cl\_base | 4 |
|  |  | 6 |
| 4 |  | инициализация указателя p\_handler на метод TYPE\_HANDLER полем p\_handler | 5 |
| 5 |  | вызов метода p\_handler с передачей переменной msg по ссылке | 6 |
| 6 |  | переход к следующему объекту | 2 |

## Алгоритм метода build\_tree\_objects\_4 класса cl\_application

Функционал: метод построения исходного дерева иерархии объектов (конструированиe моделируемой системы).

Параметры: отсутствуют.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода build\_tree\_objects\_4 класса cl\_application

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 |  | инициализация указателя на родителя base = this | 2 |
| 2 |  | инициализация указателя на ребенка base\_sub = nullptr | 3 |
| 3 |  | объявление строковых переменных s\_subordinate\_name, s\_head\_name, way | 4 |
| 4 |  | ввод переменной s\_head\_name | 5 |
|  |  |  |  |

Продолжение таблицы 8

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 5 |  | вызов метода set\_s\_object\_name с параметром s\_head\_name текущего объекта | 6 |
| 6 |  | объявление целочисленной переменной i\_class | 7 |
| 7 |  | ввод переменной way | 8 |
| 8 | way не равно "endtree" | ввод переменной s\_subordinate\_name | 9 |
|  |  | 18 |
| 9 |  | ввод переменной i\_class | 10 |
| 10 |  | приравнивание base возвращенного значения метода get\_point с параметром way | 11 |
| 11 | возвращенное значение метода get\_subordinate\_objects с параметром s\_subordinate\_name не равно nullptr |  | 12 |
|  |  | 17 |
| 12 | i\_class равно 2 | создание объекта класса cl\_2 | 17 |
|  |  | 13 |
| 13 | i\_class равно 3 | создание объекта класса cl\_3 | 17 |
|  |  | 14 |
| 14 | i\_class равно 4 | создание объекта класса cl\_4 | 17 |
|  |  | 15 |
| 15 | i\_class равно 5 | создание объекта класса cl\_5 | 17 |
|  |  | 16 |
| 16 | i\_class равно 6 | создание объекта класса cl\_6 | 17 |
|  |  | 17 |
| 17 |  | ввод переменной way | 8 |
| 18 |  | объявление строковых переменных signal\_way, handler\_way | 19 |
| 19 |  | объявление указателей на базовый класс p\_signal, p\_handler | 20 |
| 20 |  | ввод переменной signal\_way | 21 |
| 21 | signal\_way не равно "end\_of\_connections" | ввод переменной handler\_way | 22 |
|  |  | ∅ |
| 22 |  | p\_signal равно возвращенному значению метода get\_point с параметром signal\_way | 23 |
| 23 |  | p\_handler равно возвращенному значению метода get\_point с параметром handler\_way | 24 |
| 24 | p\_signal равно nullptr | вывод переноса строки, строки "Object", переменной signal\_way и строки "not found" | 27 |
|  |  | 25 |

Продолжение таблицы 8

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 25 | p\_handler равно nullptr | вывод переноса строки, строки "Handler object", переменной handler\_way и строки "not found" | 27 |
|  |  | 26 |
| 26 |  | вызов метода set\_connect с передачей параметров возвращенное значение метода get\_signal с параметром cl\_num указателя p\_signal, переменная p\_handler и возвращенное значение метода get\_handler с параметром cl\_num указателя p\_handler от указателя p\_signal | 27 |
| 27 |  | ввод переменной signal\_way | 21 |

## Алгоритм метода exec\_app\_4 класса cl\_application

Функционал: метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Параметры: отсутствуют.

Возвращаемое значение: целое.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода exec\_app\_4 класса cl\_application

| **№** | **Предикат** | **Действия** | **№ перехода** |
| --- | --- | --- | --- |
| 1 |  | вывод строки "Object tree" | 2 |
| 2 |  | вызов метода print\_from\_current\_3() | 3 |
| 3 |  | вызов метода set\_ready\_for\_tree с параметром 1 | 4 |
| 4 |  | объявление строковых переменных command, way, signal\_way, handler\_way, msg | 5 |
| 5 |  | объявление указателей на базовый класс p\_signal, p\_handler, p\_obj | 6 |
| 6 |  | объявление целочисленной переменной ready | 7 |
| 7 |  | ввод переменной command | 8 |
| 8 | command не равно "END" |  | 9 |
|  |  | ∅ |
| 9 | command равно "EMIT" | ввод переменной signal\_way | 10 |
|  |  | 15 |

Продолжение таблицы 9

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 10 |  | p\_signal равно возвращенному значению get\_point с параметром signal\_way | 11 |
| 11 |  | считывание строки в переменную msg | 12 |
| 12 | p\_signal равно nullptr | вывод переноса строки, строки "Object", переменной signal\_way и строки "not found" | 13 |
|  |  |  | 14 |
| 13 |  | ввод переменной command | 8 |
| 14 |  | вызов метода emit\_signal с параметрами возвращенное значение метода get\_signal с параметром cl\_num от указателя p\_signal и msg от p\_signal | 38 |
| 15 | command равно "SET\_CONNECT" | ввод переменной signal\_way | 16 |
|  |  |  | 24 |
| 16 |  | ввод переменной handler\_way | 17 |
| 17 |  | p\_signal равно возвращенному значению get\_point с параметром signal\_way | 18 |
| 18 |  | p\_handler равно возвращенному значению get\_point с параметром handler\_way | 19 |
| 19 | p\_signal равно nullptr | вывод переноса строки, строки "Object", переменной signal\_way и строки "not found" | 20 |
|  |  |  | 21 |
| 20 |  | ввод переменной command | 8 |
| 21 | p\_handler равно nullptr | вывод переноса строки, строки "Handler object", переменной handler\_way и строки "not found" | 22 |
|  |  |  | 23 |
| 22 |  | ввод переменной command | 8 |
| 23 |  | вызов метода set\_connect с передачей параметров возвращенное значение метода get\_signal с параметром cl\_num указателя p\_signal, переменная p\_handler и возвращенное значение метода get\_handler с параметром cl\_num указателя p\_handler от указателя p\_signal | 38 |
| 24 | command равно "DELETE\_CONNECT" | ввод переменной signal\_way | 25 |
|  |  |  | 33 |
| 25 |  | ввод переменной handler\_way | 26 |
| 26 |  | p\_signal равно возвращенному значению get\_point с параметром signal\_way | 27 |
| 27 |  | p\_handler равно возвращенному значению get\_point с параметром handler\_way | 28 |
| 28 | p\_signal равно nullptr | вывод переноса строки, строки "Object", переменной signal\_way и строки "not found" | 29 |
|  |  |  | 30 |
| 29 |  | ввод переменной command | 8 |

Продолжение таблицы 9

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 30 | p\_handler равно nullptr | вывод переноса строки, строки "Handler object", переменной handler\_way и строки "not found" | 31 |
|  |  |  | 32 |
| 31 |  | ввод переменной command | 8 |
| 32 |  | вызов метода delete\_connect с передачей параметров возвращенное значение метода get\_signal с параметром cl\_num указателя p\_signal, переменная p\_handler и возвращенное значение метода get\_handler с параметром cl\_num указателя p\_handler от указателя p\_signal | 38 |
| 33 | command равно "SET\_CONDITION" | ввод переменной way | 34 |
|  |  | 38 |
| 34 |  | ввод переменной ready | 35 |
| 35 |  | p\_obj равно возвращенному значению метода get\_point с параметром way | 36 |
| 36 | p\_obj равно nullptr | вывод переноса строки, строки "Object", переменной way и строки "not found" | 37 |
|  |  | 38 |
| 37 |  | ввод переменной command | 8 |
| 38 |  | вызов метода set\_reay с параметром ready от p\_obj | 39 |
| 39 |  | ввод переменной command | 8 |

# 4 Блок-схемы алгоритмов

Представим описание алгоритмов в графическом виде на рисунках 1-13.

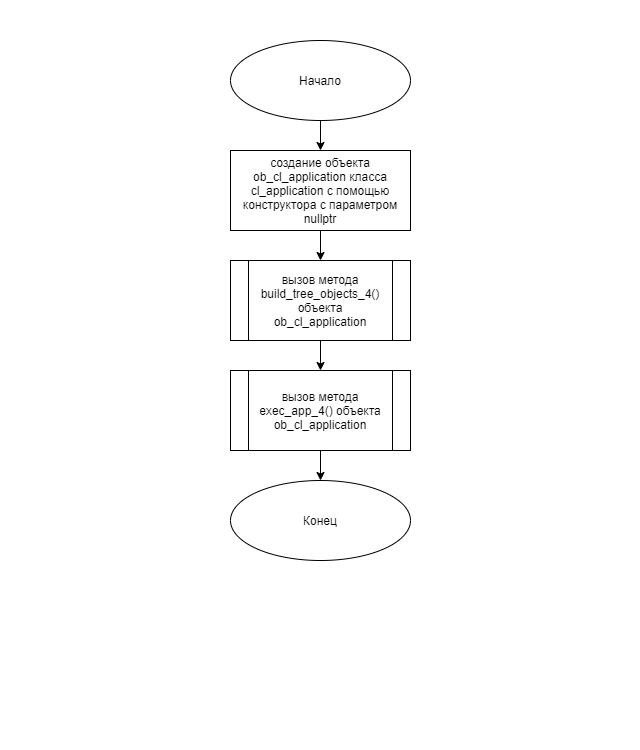


Рисунок 1 – Блок-схема алгоритма

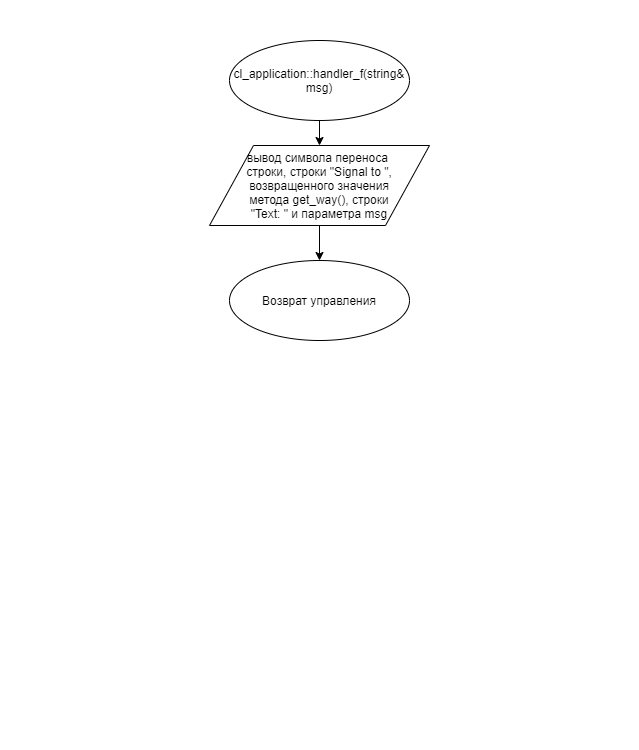


Рисунок 2 – Блок-схема алгоритма

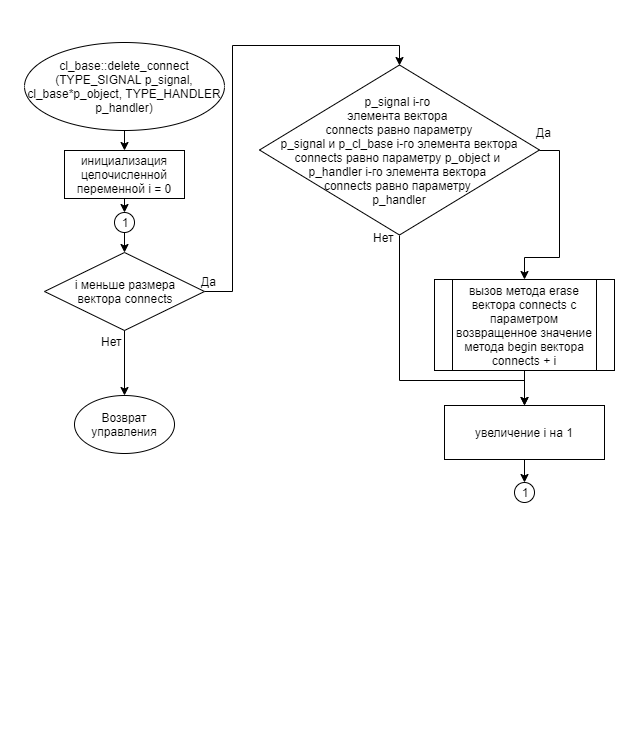


Рисунок 3 – Блок-схема алгоритма

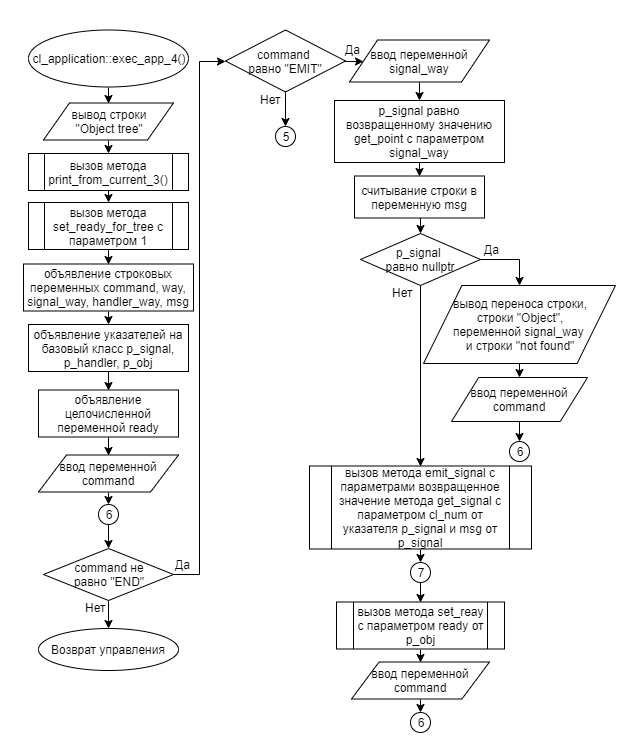


Рисунок 4 – Блок-схема алгоритма

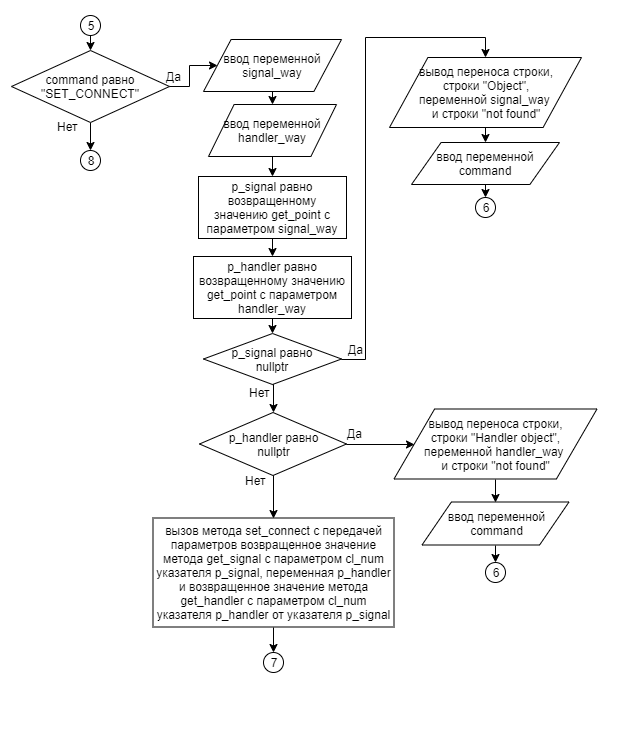


Рисунок 5 – Блок-схема алгоритма

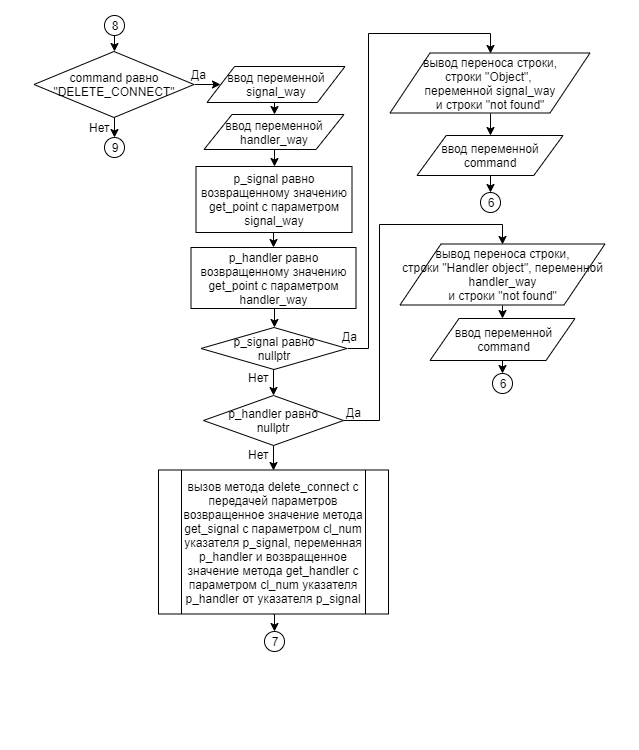


Рисунок 6 – Блок-схема алгоритма

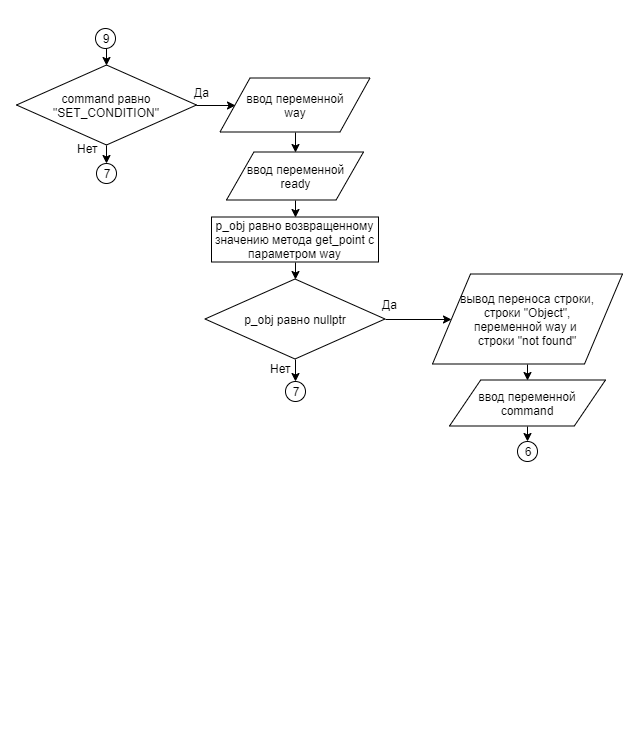


Рисунок 7 – Блок-схема алгоритма

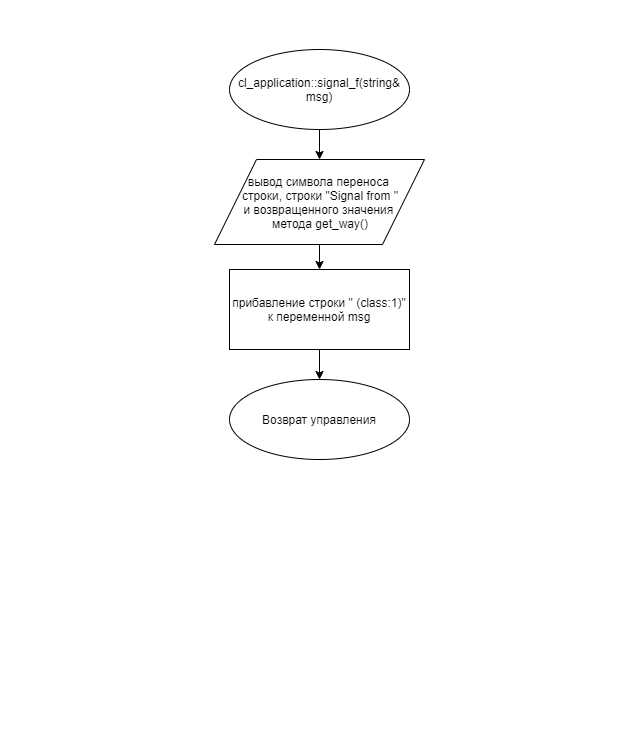


Рисунок 8 – Блок-схема алгоритма

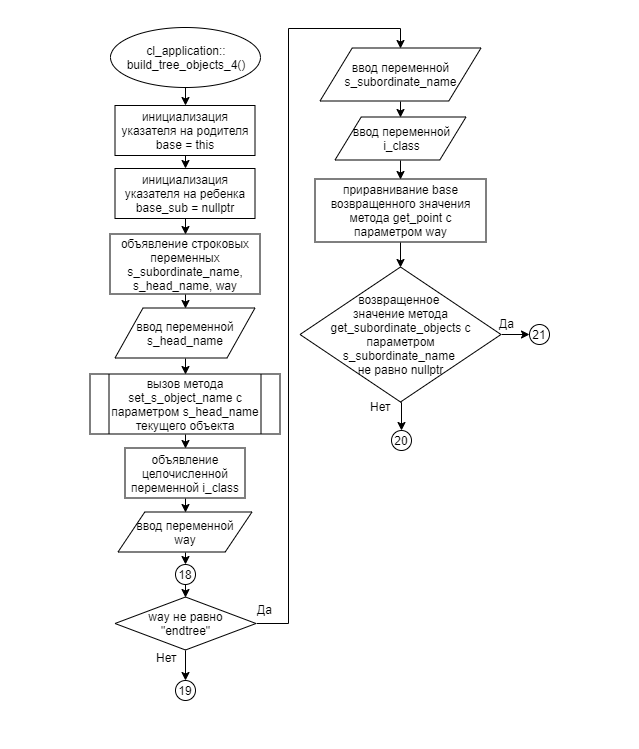


Рисунок 9 – Блок-схема алгоритма

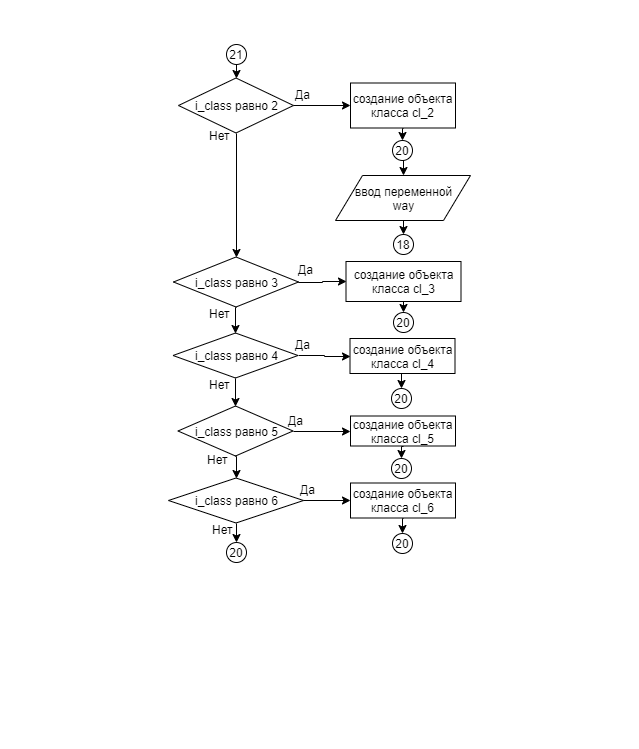


Рисунок 10 – Блок-схема алгоритма

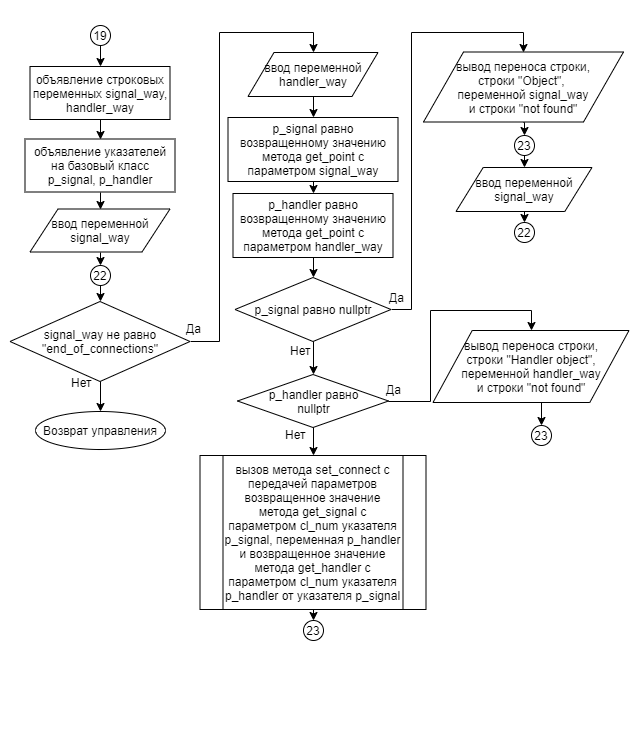


Рисунок 11 – Блок-схема алгоритма

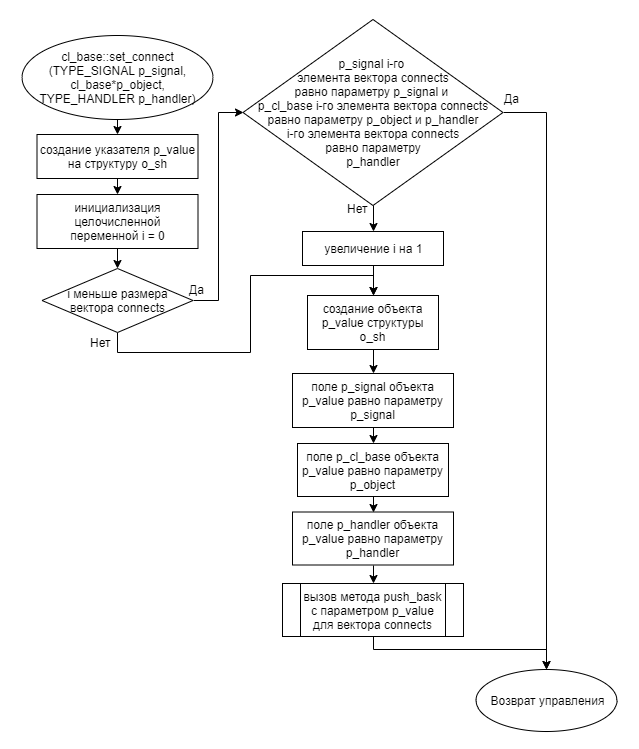


Рисунок 12 – Блок-схема алгоритма

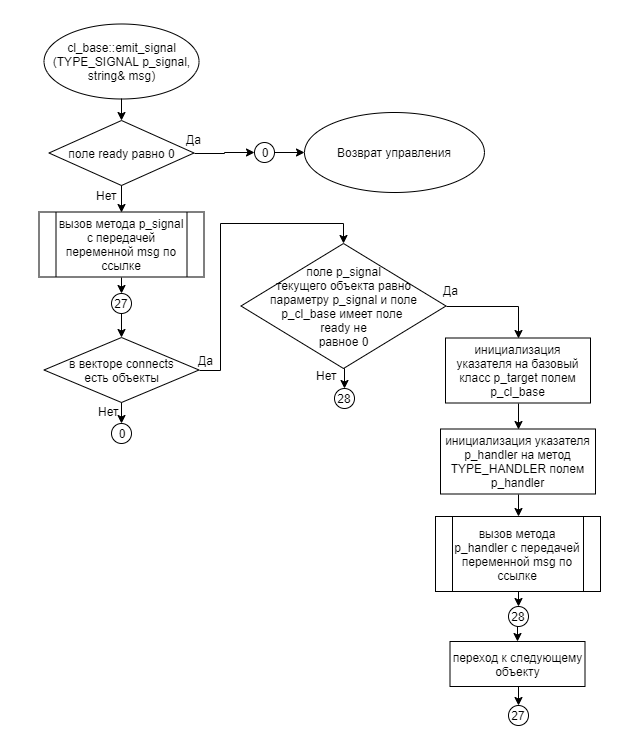


Рисунок 13 – Блок-схема алгоритма

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы был разработан и реализован механизм сигналов и обработчиков в базовом классе. Этот механизм предоставляет удобный способ организации взаимодействия объектов, позволяя объектам инициировать сигналы и передавать их нескольким обработчикам. Реализованные методы позволяют установить и удалить связь между сигналом объекта и обработчиком, а также выдать сигнал с передачей данных.

Механизм сигналов и обработчиков имеет широкий спектр применений и может быть использован в различных областях программирования. Он упрощает взаимодействие между объектами, обеспечивая гибкость и модульность. Разработанный механизм предоставляет гибкую архитектуру, полезную при создании сложных систем и приложений.

Применение механизма сигналов и обработчиков позволяет эффективно организовывать взаимодействие объектов, упрощает обмен информацией и повышает переиспользуемость кода. Он предоставляет удобный способ передачи сигналов и данных между объектами, а также гибкое управление взаимодействием в рамках системы.

В итоге, разработанный механизм сигналов и обработчиков является мощным инструментом для организации взаимодействия объектов, способствуя созданию более эффективных, модульных и переиспользуемых систем программного обеспечения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения 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).

3. 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).

4. Шилдт Г. С++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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

# Приложение 1. Код программы

## Файл cl\_2.cpp

Листинг 1 – cl\_2.cpp

|  |
| --- |
| #include "cl\_2.h"  cl\_2::cl\_2(cl\_base \* p\_head\_object, string s\_object\_name): cl\_base(p\_head\_object, s\_object\_name){ //вызов конструктора cl\_base с передачей параметров p\_head\_object, s\_object\_name  cl\_num = 2; //поле cl\_num равно 2  };  void cl\_2::signal\_f(string& msg){  cout << "\nSignal from " << get\_way(); //вывод символа переноса строки, строки "Signal from " и возвращенного значения метода get\_way()  msg += " (class: 2)"; //прибавление строки " (class:2)" к переменной msg  }  void cl\_2::handler\_f(string& msg){  cout << "\nSignal to " << get\_way() << " Text: " << msg; // вывод символа переноса строки, строки "Signal to ", возвращенного значения метода get\_way(), строки "Text: " и параметра msg  } |

## Файл cl\_2.h

Листинг 2 – cl\_2.h

|  |
| --- |
| #ifndef CL\_2\_H  #define CL\_2\_H  #include "cl\_base.h"  class cl\_2 : public cl\_base{  public:  cl\_2(cl\_base \*p\_head\_object, string s\_object\_name); //конструктор, устанавливает значения скрытых свойств  void signal\_f(string& msg); //метод сигнала  void handler\_f(string& msg); //метод обработчика  };  #endif |

## Файл cl\_3.cpp

Листинг 3 – cl\_3.cpp

|  |
| --- |
| #include "cl\_3.h"  cl\_3::cl\_3(cl\_base \* p\_head\_object, string s\_object\_name): cl\_base(p\_head\_object, s\_object\_name){ //вызов конструктора cl\_base с передачей параметров p\_head\_object, s\_object\_name  cl\_num = 3; //поле cl\_num равно 3  };  void cl\_3::signal\_f(string& msg){  cout << "\nSignal from " << get\_way(); //вывод символа переноса строки, строки "Signal from " и возвращенного значения метода get\_way()  msg += " (class: 3)"; //прибавление строки " (class:3)" к переменной msg  }  void cl\_3::handler\_f(string& msg){  cout << "\nSignal to " << get\_way() << " Text: " << msg; // вывод символа переноса строки, строки "Signal to ", возвращенного значения метода get\_way(), строки "Text: " и параметра msg  } |

## Файл cl\_3.h

Листинг 4 – cl\_3.h

|  |
| --- |
| #ifndef CL\_3\_H  #define CL\_3\_H  #include "cl\_base.h"  class cl\_3 : public cl\_base{  public:  cl\_3(cl\_base \*p\_head\_object, string s\_object\_name); //конструктор, устанавливает значения скрытых свойств  void signal\_f(string& msg); //метод сигнала  void handler\_f(string& msg); //метод обработчика  };  #endif |

## Файл cl\_4.cpp

Листинг 5 – cl\_4.cpp

|  |
| --- |
| #include "cl\_4.h"  cl\_4::cl\_4(cl\_base \* p\_head\_object, string s\_object\_name): cl\_base(p\_head\_object, s\_object\_name){ //вызов конструктора cl\_base с передачей параметров p\_head\_object, s\_object\_name  cl\_num = 4; //поле cl\_num равно 4  };  void cl\_4::signal\_f(string& msg){  cout << "\nSignal from " << get\_way(); //вывод символа переноса строки, строки "Signal from " и возвращенного значения метода get\_way()  msg += " (class: 4)"; //прибавление строки " (class:4)" к переменной msg  }  void cl\_4::handler\_f(string& msg){  cout << "\nSignal to " << get\_way() << " Text: " << msg; // вывод символа переноса строки, строки "Signal to ", возвращенного значения метода get\_way(), строки "Text: " и параметра msg  } |

## Файл cl\_4.h

Листинг 6 – cl\_4.h

|  |
| --- |
| #ifndef \_\_CL\_4\_\_H  #define \_\_CL\_4\_\_H  #include "cl\_base.h"  class cl\_4 : public cl\_base{  public:  cl\_4(cl\_base \*p\_head\_object, string s\_object\_name); //конструктор, устанавливает значения скрытых свойств  void signal\_f(string& msg); //метод сигнала  void handler\_f(string& msg); //метод обработчика  };  #endif |

## Файл cl\_5.cpp

Листинг 7 – cl\_5.cpp

|  |
| --- |
| #include "cl\_5.h"  cl\_5::cl\_5(cl\_base \* p\_head\_object, string s\_object\_name): cl\_base(p\_head\_object, s\_object\_name){ //вызов конструктора cl\_base с передачей параметров p\_head\_object, s\_object\_name  cl\_num = 5; //поле cl\_num равно 5  };  void cl\_5::signal\_f(string& msg){  cout << "\nSignal from " << get\_way(); //вывод символа переноса строки, строки "Signal from " и возвращенного значения метода get\_way()  msg += " (class: 5)"; //прибавление строки " (class:5)" к переменной msg  }  void cl\_5::handler\_f(string& msg){  cout << "\nSignal to " << get\_way() << " Text: " << msg; // вывод символа переноса строки, строки "Signal to ", возвращенного значения метода get\_way(), строки "Text: " и параметра msg  } |

## Файл cl\_5.h

Листинг 8 – cl\_5.h

|  |
| --- |
| #ifndef \_\_CL\_5\_\_H  #define \_\_CL\_5\_\_H  #include "cl\_base.h"  class cl\_5 : public cl\_base{  public:  cl\_5(cl\_base \*p\_head\_object, string s\_object\_name); //конструктор, устанавливает значения скрытых свойств  void signal\_f(string& msg); //метод сигнала  void handler\_f(string& msg); //метод обработчика  };  #endif |

## Файл cl\_6.cpp

Листинг 9 – cl\_6.cpp

|  |
| --- |
| #include "cl\_6.h"  cl\_6::cl\_6(cl\_base \* p\_head\_object, string s\_object\_name): cl\_base(p\_head\_object, s\_object\_name){ //вызов конструктора cl\_base с передачей параметров p\_head\_object, s\_object\_name  cl\_num = 6; //поле cl\_num равно 6  };  void cl\_6::signal\_f(string& msg){  cout << "\nSignal from " << get\_way(); //вывод символа переноса строки, строки "Signal from " и возвращенного значения метода get\_way()  msg += " (class: 6)"; //прибавление строки " (class:6)" к переменной msg  }  void cl\_6::handler\_f(string& msg){  cout << "\nSignal to " << get\_way() << " Text: " << msg; // вывод символа переноса строки, строки "Signal to ", возвращенного значения метода get\_way(), строки "Text: " и параметра msg  } |

## Файл cl\_6.h

Листинг 20 – cl\_6.h

|  |
| --- |
| #ifndef \_\_CL\_6\_\_H  #define \_\_CL\_6\_\_H  #include "cl\_base.h"  class cl\_6 : public cl\_base{  public:  cl\_6(cl\_base \*p\_head\_object, string s\_object\_name); //конструктор, устанавливает значения скрытых свойств  void signal\_f(string& msg); //метод сигнала  void handler\_f(string& msg); //метод обработчика  };  #endif |

## Файл cl\_application.cpp

Листинг 13 – cl\_application.cpp

|  |
| --- |
| #include "cl\_application.h"  #include "cl\_2.h"  #include "cl\_3.h"  #include "cl\_4.h"  #include "cl\_5.h"  #include "cl\_6.h"  cl\_application::cl\_application(cl\_base\* p\_head\_object):cl\_base(p\_head\_object){ //вызов конструктора cl\_base с передачей параметра p\_head\_object  cl\_num = 1; //поле cl\_num равно 1  }  void cl\_application::signal\_f(string& msg){  cout << "\nSignal from " << get\_way(); //вывод символа переноса строки, строки "Signal from " и возвращенного значения метода get\_way()  msg += " (class: 1)"; //прибавление строки " (class:1)" к переменной msg  }  void cl\_application::handler\_f(string& msg){  cout << "\nSignal to " << get\_way() << " Text: " << msg; // вывод символа переноса строки, строки "Signal to ", возвращенного значения метода get\_way(), строки "Text: " и параметра msg  }  TYPE\_SIGNAL cl\_application::get\_signal(int n){  switch(n){  case 1:  return SIGNAL\_D(cl\_application::signal\_f); //возврат значения функции SIGNAL\_D с передачей метода signal\_f класса cl\_application  break;  case 2:  return SIGNAL\_D(cl\_2::signal\_f); //возврат значения функции SIGNAL\_D с передачей метода signal\_f класса cl\_2  break;  case 3:  return SIGNAL\_D(cl\_3::signal\_f); //возврат значения функции SIGNAL\_D с передачей метода signal\_f класса cl\_3  break;  case 4:  return SIGNAL\_D(cl\_4::signal\_f); //возврат значения функции SIGNAL\_D с передачей метода signal\_f класса cl\_4  break;  case 5:  return SIGNAL\_D(cl\_5::signal\_f); //возврат значения функции SIGNAL\_D с передачей метода signal\_f класса cl\_5  break;  case 6:  return SIGNAL\_D(cl\_6::signal\_f); //возврат значения функции SIGNAL\_D с передачей метода signal\_f класса cl\_6  break;  default:  return nullptr; //возврат nullptr  }  }  TYPE\_HANDLER cl\_application::get\_handler(int n){  switch(n){  case 1:  return HANDLER\_D(cl\_application::handler\_f); //возврат значения функции HANDLER\_D с передачей метода handler\_f класса cl\_application  break;  case 2:  return HANDLER\_D(cl\_2::handler\_f); //возврат значения функции HANDLER\_D с передачей метода handler\_f класса cl\_2  break;  case 3:  return HANDLER\_D(cl\_3::handler\_f); //возврат значения функции HANDLER\_D с передачей метода handler\_f класса cl\_3  break;  case 4:  return HANDLER\_D(cl\_4::handler\_f); //возврат значения функции HANDLER\_D с передачей метода handler\_f класса cl\_4  break;  case 5:  return HANDLER\_D(cl\_5::handler\_f); //возврат значения функции HANDLER\_D с передачей метода handler\_f класса cl\_5  break;  case 6:  return HANDLER\_D(cl\_6::handler\_f); //возврат значения функции HANDLER\_D с передачей метода handler\_f класса cl\_6  break;  default:  return nullptr; //возврат nullptr  }  }  void cl\_application::build\_tree\_objects\_4(){  cl\_base \* base = this; //инициализация указателя на родителя base = this  cl\_base \* base\_sub = nullptr; //инициализация указателя на ребенка base\_sub = nullptr  string s\_subordinate\_name, s\_head\_name, way; //объявление переменных строкового типа s\_subordinate\_name, s\_head\_name и way  cin >> s\_head\_name; //ввод переменной s\_head\_name  this -> set\_s\_object\_name(s\_head\_name); //вызов метода set\_object\_name(s\_head\_name)  int i\_class; //объявление целочисленной переменной i\_class  cin >> way; //ввод переменной way  while (!(way == "endtree")){ //way не равно "endtree"  cin >> s\_subordinate\_name; //ввод переменной s\_subordinate\_name  cin >> i\_class; //ввод переменной i\_class  base = get\_point(way); //присваивание base возвращенного значения метода get\_point(way)  if (!base -> get\_subordinate\_objects(s\_subordinate\_name)){ //возвращенное значение метода get\_subordinate\_objects с параметром s\_subordinate\_name не равно nullptr  switch(i\_class){  case 2:  base\_sub = new cl\_2(base, s\_subordinate\_name); //создание объекта класса cl\_2  break;  case 3:  base\_sub = new cl\_3(base, s\_subordinate\_name); //создание объекта класса cl\_3  break;  case 4:  base\_sub = new cl\_4(base, s\_subordinate\_name); //создание объекта класса cl\_4  break;  case 5:  base\_sub = new cl\_5(base, s\_subordinate\_name); //создание объекта класса cl\_5  break;  case 6:  base\_sub = new cl\_6(base, s\_subordinate\_name); //создание объекта класса cl\_6  break;  }  }  cin >> way; //ввод переменной way  }  string signal\_way, handler\_way; //объявление строковых переменных signal\_way, handler\_way  cl\_base \*p\_signal, \*p\_handler; //объявление указателей на базовый класс p\_signal, p\_handler  cin >> signal\_way; //ввод переменной signal\_way  while (!(signal\_way == "end\_of\_connections")){ //signal\_way не равно "end\_of\_connections"  cin >> handler\_way; //ввод переменной handler\_way  p\_signal = get\_point(signal\_way); //p\_signal равно возвращенному значению метода get\_point с параметром signal\_way  p\_handler = get\_point(handler\_way); //p\_handler равно возвращенному значению метода get\_point с параметром handler\_way  if (!p\_signal){ //p\_signal равно nullptr  cout << "\nObject " << signal\_way << " not found"; //вывод переноса строки, строки "Object", переменной signal\_way и строки "not found"  continue;  }  if (!p\_handler){ // p\_handler равно nullptr  cout << "\nHandler object " << handler\_way << " not found"; //вывод переноса строки, строки "Handler object", переменной handler\_way и строки "not found"  continue;  }  p\_signal -> set\_connect(get\_signal(p\_signal -> cl\_num), p\_handler, get\_handler(p\_handler -> cl\_num)); //вызов метода set\_connect с передачей параметров возвращенное значение метода get\_signal с параметром cl\_num указателя p\_signal, переменная p\_handler и возвращенное значение метода get\_handler с параметром cl\_num указателя p\_handler от указателя p\_signal  cin >> signal\_way; //ввод переменной signal\_way  }  }  int cl\_application::exec\_app\_4(){  cout << "Object tree"; //вывод строки "Object tree"  print\_from\_current\_3(); //вызов метода print\_from\_current\_3()  set\_ready\_for\_tree(1); //вызов метода set\_ready\_for\_tree с параметром 1  string command, way, signal\_way, handler\_way, msg; //объявление строковых переменных command, way, signal\_way, handler\_way, msg  cl\_base \*p\_signal, \*p\_handler, \*p\_obj; //объявление указателей на базовый класс p\_signal, p\_handler, p\_obj  int ready; //объявление целочисленной переменной ready  cin >> command; //ввод переменной command  while (command != "END"){ //command не равно "END"  if (command == "EMIT"){ //command равно "EMIT"  cin >> signal\_way; //ввод переменной signal\_way  p\_signal = get\_point(signal\_way); //p\_signal равно возвращенному значению get\_point с параметром signal\_way  getline(cin, msg); //считывание строки в переменную msg  if (!p\_signal){ //p\_signal равно nullptr  cout << "\nObject " << signal\_way << " not found"; //вывод переноса строки, строки "Object", переменной signal\_way и строки "not found"  cin >> command; //ввод переменной command  continue;  }  p\_signal -> emit\_signal(get\_signal(p\_signal -> cl\_num), msg); //вызов метода emit\_signal с параметрами возвращенное значение метода get\_signal с параметром cl\_num от указателя p\_signal и msg от p\_signal  } else if (command == "SET\_CONNECT"){ //command равно "SET\_CONNECT"  cin >> signal\_way; //ввод переменной signal\_way  cin >> handler\_way; //ввод переменной handler\_way  p\_signal = get\_point(signal\_way); //p\_signal равно возвращенному значению get\_point с параметром signal\_way  p\_handler = get\_point(handler\_way); //p\_handler равно возвращенному значению get\_point с параметром handler\_way  if (!p\_signal){ //p\_signal равно nullptr  cout << "\nObject " << signal\_way << " not found"; //вывод переноса строки, строки "Object", переменной signal\_way и строки "not found"  cin >> command; //ввод переменной command  continue;  }  if (!p\_handler){ //p\_handler равно nullptr  cout << "\nHandler object " << handler\_way << " not found"; //вывод переноса строки, строки "Handler object", переменной handler\_way и строки "not found"  cin >> command; //ввод переменной command  continue;  }  p\_signal -> set\_connect(get\_signal(p\_signal -> cl\_num), p\_handler, get\_handler(p\_handler -> cl\_num)); //вызов метода set\_connect с передачей параметров возвращенное значение метода get\_signal с параметром cl\_num указателя p\_signal, переменная p\_handler и возвращенное значение метода get\_handler с параметром cl\_num указателя p\_handler от указателя p\_signal  } else if (command == "DELETE\_CONNECT"){ //command равно "DELETE\_CONNECT"  cin >> signal\_way; //ввод переменной signal\_way  cin >> handler\_way; //ввод переменной handler\_way  p\_signal = get\_point(signal\_way); //p\_signal равно возвращенному значению get\_point с параметром signal\_way  p\_handler = get\_point(handler\_way); //p\_handler равно возвращенному значению get\_point с параметром handler\_way  if (!p\_signal){ //p\_signal равно nullptr  cout << "\nObject " << signal\_way << " not found"; //вывод переноса строки, строки "Object", переменной signal\_way и строки "not found"  cin >> command; //ввод переменной command  continue;  }  if (!p\_handler){ //p\_handler равно nullptr  cout << "\nHandler object " << handler\_way << " not found"; //вывод переноса строки, строки "Handler object", переменной handler\_way и строки "not found"  cin >> command; //ввод переменной command  continue;  }  p\_signal -> delete\_connect(get\_signal(p\_signal -> cl\_num), p\_handler, get\_handler(p\_handler -> cl\_num)); //вызов метода delete\_connect с передачей параметров возвращенное значение метода get\_signal с параметром cl\_num указателя p\_signal, переменная p\_handler и возвращенное значение метода get\_handler с параметром cl\_num указателя p\_handler от указателя p\_signal  } else if (command == "SET\_CONDITION"){ //command равно "SET\_CONDITION"  cin >> way; //ввод переменной way  cin >> ready; //ввод переменной ready  p\_obj = get\_point(way); //p\_obj равно возвращенному значению метода get\_point с параметром way  if (!p\_obj){ //p\_obj равно nullptr  cout << "\nObject " << way << " not found"; //вывод переноса строки, строки "Object", переменной way и строки "not found"  cin >> command; //ввод переменной command  continue;  }  p\_obj -> set\_ready(ready); //вызов метода set\_reay с параметром ready от p\_obj  }  cin >> command; //ввод переменной command  }  return 0;  } |

## Файл cl\_application.h

Листинг 42 – cl\_application.h

|  |
| --- |
| #ifndef CL\_APPLICATION\_H  #define CL\_APPLICATION\_H  #include "cl\_base.h"  class cl\_application : public cl\_base{  public:  cl\_application(cl\_base\* p\_head\_object); //конструктор, устанавливает значения скрытых свойств  void signal\_f(string& msg); //метод сигнала  void handler\_f(string& msg); //метод обработчика  TYPE\_SIGNAL get\_signal(int n); //выдача сигнала по номеру класса  TYPE\_HANDLER get\_handler(int n); //выдача метода обработчика по номеру класса  void build\_tree\_objects\_4();//метод построения исходного дерева иерархии объектов (конструированиe моделируемой системы)  int exec\_app\_4();//метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи)  };  #endif |

## Файл cl\_base.cpp

Листинг 53 – cl\_base.cpp

|  |
| --- |
| #include "cl\_base.h"  cl\_base::cl\_base(cl\_base \* p\_head\_object, string s\_object\_name){  this -> p\_head\_object = p\_head\_object; //присвоение полю p\_head\_object значения параметра p\_head\_object  this -> s\_object\_name = s\_object\_name; //присвоение полю s\_object\_name значения параметра s\_object\_name  if (get\_p\_head\_object()){ //возвращенное значение метода get\_p\_head\_object не равно 0  p\_head\_object -> subordinate\_objects.push\_back(this); //добавление адреса текущего объекта с помощью метода push\_back в вектор subordinate\_objects объекта по указателю p\_head\_object  }  }  string cl\_base::get\_s\_object\_name(){  return s\_object\_name; //возврат поля s\_object\_name  }  cl\_base \* cl\_base::get\_p\_head\_object(){  return p\_head\_object; //возврат поля p\_head\_object  }  cl\_base \* cl\_base::get\_subordinate\_objects(string s\_name){  for (int i = 0; i < subordinate\_objects.size(); i++){ //i меньше размера вектора subordinate\_objects  if (subordinate\_objects[i] -> get\_s\_object\_name() == s\_name){ //возвращенное значение метода get\_s\_object\_name от i-го элемента вектора subordinate\_objects равно значению параметра s\_name  return subordinate\_objects[i]; //возврат i-го элемента вектора subordinate\_objects  }  }  return nullptr; //возврат nullptr  }  bool cl\_base::set\_s\_object\_name(string s\_new\_name){  if (get\_p\_head\_object()){ //возвращаемое значение метода get\_p\_head\_object не равно 0  for (int i = 0; i < get\_p\_head\_object() -> subordinate\_objects.size(); i++){ //i меньше размера вектора subordinate\_objects  if (get\_p\_head\_object() -> subordinate\_objects[i] -> get\_s\_object\_name() == s\_new\_name){return false;} //возвращенное значение метода get\_p\_head\_object от i-го элемента вектора subordinate\_objects равно значению параметра s\_new\_name  }  }  this -> s\_object\_name = s\_new\_name; //поле s\_object\_name равно значению параметра s\_new\_name  return true;  }  cl\_base::~cl\_base(){  for (int i = 0; i < subordinate\_objects.size(); i++){delete subordinate\_objects[i];} // i меньше размера вектора subordinate\_objects, удаление i-го элемента вектора subordinate\_objects  }  int cl\_base::count(string s\_name){  int cnt = 0; //инициализация целочисленной переменной cnt = 0  if (this -> get\_s\_object\_name() == s\_name){cnt++;} //возвращенное значение метода get\_s\_object\_name от текущего объекта равно параметру s\_name, увеличение cnt на 1  for (auto p\_sub\_object : subordinate\_objects){ //в векторе subordinate\_objects есть объекты  cnt += p\_sub\_object -> count(s\_name); //прибавление к переменной cnt возвращенного значения метода count с параметром s\_name от p\_sub\_object  }  return cnt; //возврат значения переменной cnt  }  cl\_base\* cl\_base::search\_object(string s\_name){  if (this -> get\_s\_object\_name() == s\_name){return this;} //возвращенное значение метода get\_s\_object\_name от текущего объекта равно параметру s\_name  for (auto p\_sub\_object : subordinate\_objects){ //в векторе subordinate\_objects есть объекты  cl\_base\* p\_found = p\_sub\_object -> search\_object(s\_name); //присвоение указателю p\_found возвращенного значения метода search\_object с параметром s\_name от p\_sub\_object  if (p\_found != nullptr){return p\_found;} //указатель p\_found не равен nullptr, возврат указателя p\_found  }  return nullptr; //возврат nullptr  }  cl\_base\* cl\_base::find\_object\_from\_current(string s\_name){  if (count(s\_name) != 1){return nullptr;} //возвращенное значение метода count с параметром s\_name не равно 1  return search\_object(s\_name); //возврат возвращенного значения метода search\_object с параметром s\_name  }  cl\_base\* cl\_base::find\_root\_from\_current(){  if (this -> get\_p\_head\_object() == nullptr){return this;} //возвращенное значение метода get\_p\_head\_object от текущего объекта равно nullptr  return get\_p\_head\_object() -> find\_root\_from\_current(); //возврат возвращенного значения метода find\_root\_from\_current от возвращенного значения метода get\_p\_head\_object  }  cl\_base\* cl\_base::find\_object\_from\_root(string s\_name){  return find\_root\_from\_current() -> find\_object\_from\_current(s\_name); //возврат возвращенного значения метода find\_object\_from\_current с параметром s\_name от возвращенного значения метода find\_root\_from\_current  }  int cl\_base::get\_ready(){  return this -> ready; //возврат поля ready от текущего объекта  }  void cl\_base::set\_ready(int i\_ready){  if (i\_ready != 0){ //параметра i\_ready не равен 0  cl\_base\* p = this -> get\_p\_head\_object(); //инициализация указателя p возвращенным значением метода get\_p\_head\_object от текущего объекта  while (p){ //указатель p не равен nullptr  if (p -> get\_ready() == 0){return;} //возвращенное значение метода get\_ready от указателя р равно 0  p = p -> get\_p\_head\_object(); //присвоение указателю р возвращенного значения метода get\_p\_head\_object от указателя р  }  this -> ready = i\_ready; //поле ready равно значению параметра i\_ready  } else {  this -> ready = i\_ready; //поле ready равно значению параметра i\_ready  for (auto p\_sub\_object : subordinate\_objects){p\_sub\_object -> set\_ready(i\_ready);} //вызов метода set\_ready с параметром i\_ready от p\_sub\_object  }  }  cl\_base\* cl\_base::get\_point(string way){  if (way == ""){return nullptr;} //way равно "", возврат nullptr  else if (way == "/"){return find\_root\_from\_current();} //way равно "/", возврат возвращенного значения метода find\_root\_from\_current()  else if (way == "."){return this;} //way равно ".", возврат указателя this  else if ((way[0] == '/') && (way[1] == '/')){return find\_object\_from\_root(way.substr(2));} //0-ой символ переменной way равен '/' и 1-ый символ переменной way равен '/', возврат возвращенного значения метода find\_object\_from\_root с параметром возвращенное подстроки way со 2 элемента с помощью метода substr  else if (way[0] == '.'){return find\_object\_from\_current(way.substr(1));} //0-ой символ переменной way равен '.', возврат возвращенного значения метода find\_object\_from\_root с параметром возвращенной подстроки way с 1 элемента с помощью метода substr  else if (way[0] == '/'){ //0-ой символ переменной way равен '/'  cl\_base\* p = find\_root\_from\_current(); //инициализация указателя на базовый класс р возвращенным знаечнием метода find\_root\_from\_current()  string name = ""; //инициализация строковой переменной name = ""  way = way.substr(1); //присвоение переменной way возвращенного значения метода substr(1) для way  for (char ch : way){ // в строке way есть символы ch  if (ch == '/'){ //ch равно '/'  p = p -> get\_subordinate\_objects(name); //присвоение указателю р возвращенного значения метода get\_subordinate\_obejcts с параметром name от указателя р  if (!p) {return p;} //р равно nullptr, возврат р  name = ""; //присвоение переменной name ""  }else{name += ch;} //присвоение переменной name значения name + ch  }  p = p -> get\_subordinate\_objects(name); //присвоение указателю р возвращенного значения метода get\_subordinate\_obejcts с параметром name от указателя р  return p; //возврат р  }else if(way[0] != '/'){ //0-ой символ переменной way не равен '/'  cl\_base\* p = this; //инициализация указателя на базовый класс р = this  string name = ""; //инициализация строковой переменной name = ""  for (char ch : way){ // в строке way есть символы ch  if (ch == '/'){ //ch равно '/'  p = p -> get\_subordinate\_objects(name); //присвоение указателю p возвращенного значения метода get\_subordinate\_objects(name)  if (!p) {return p;} //р равно nullptr, возврат р  name = ""; //присвоение переменной name ""  }else{name += ch;} //присвоение переменной name значения name + ch  }  p = p -> get\_subordinate\_objects(name); //присвоение указателю р возвращенного значения метода get\_subordinate\_obejcts с параметром name от указателя р  return p; //возврат р  }else{return nullptr;} //возврат nullptr  }  void cl\_base::print\_from\_current\_3(){  int tab = 0; //инициализация целочисленной переменной tab = 0  cl\_base\* p = this -> get\_p\_head\_object(); //инициализация указателя р на базовый класс cl\_base возвращенным значением метода get\_p\_head\_object от текущего объекта  while (p){ //р не равно nullptr  p = p -> get\_p\_head\_object(); //присвоение указателю р возвращенного значения метода get\_p\_head\_object от указателя р  tab++; //увеличение tab на 1  }  cout << endl << string(tab\*4, ' ') << this -> get\_s\_object\_name(); //вывод переноса строки, строки tab\*4, ' ' и возвращенного значения метода get\_s\_object\_name от текущего объекта  for (auto p\_sub\_object : subordinate\_objects){p\_sub\_object -> print\_from\_current\_3();} //в векторе subordinate\_obejcts есть объекты, вызов метода print\_from\_current\_3 от p\_sub\_object  }  void cl\_base::set\_connect(TYPE\_SIGNAL p\_signal, cl\_base\* p\_object, TYPE\_HANDLER p\_handler){  o\_sh\* p\_value; //создание указателя p\_value на структуру o\_sh  for (int i = 0; i < connects.size(); i++){  if (connects[i] -> p\_signal == p\_signal && connects[i] -> p\_cl\_base == p\_object && connects[i] -> p\_handler == p\_handler){return;} //p\_signal i-го элемента вектора connects равно параметру p\_signal и p\_cl\_base i-го элемента вектора connects равно параметру p\_object и p\_handler i-го элемента вектора connects равно параметру p\_handler  }  p\_value = new o\_sh(); //создание объекта p\_value структуры o\_sh  p\_value -> p\_signal = p\_signal; //поле p\_signal объекта p\_value равно параметру p\_signal  p\_value -> p\_cl\_base = p\_object; //поле p\_cl\_base объекта p\_value равно параметру p\_object  p\_value -> p\_handler = p\_handler; //поле p\_handler объекта p\_value равно параметру p\_handler  connects.push\_back(p\_value); //вызов метода push\_bask с параметром p\_value для вектора connects  }  void cl\_base::delete\_connect(TYPE\_SIGNAL p\_signal, cl\_base\* p\_object, TYPE\_HANDLER p\_handler){  for (int i = 0; i < connects.size(); i++){  if (connects[i] -> p\_signal == p\_signal && connects[i] -> p\_cl\_base == p\_object && connects[i] -> p\_handler == p\_handler){ //p\_signal i-го элемента вектора connects равно параметру p\_signal и p\_cl\_base i-го элемента вектора connects равно параметру p\_object и p\_handler i-го элемента вектора connects равно параметру p\_handler  connects.erase(connects.begin() + i); //вызов метода erase вектора connects с параметром возвращенное значение метода begin вектора connects + i  }  }  }  void cl\_base::emit\_signal(TYPE\_SIGNAL p\_signal, string& msg){  if (!ready){return;} //поле ready равно 0  (this ->\* p\_signal)(msg); //вызов метода p\_signal с передачей переменной msg по ссылке  for (auto connection : connects){ //в векторе connects есть объекты  if (connection -> p\_signal == p\_signal && connection -> p\_cl\_base -> ready){ //поле p\_signal текущего объекта равно параметру p\_signal и поле p\_cl\_base имеет поле ready не равное 0  cl\_base\* p\_target = connection -> p\_cl\_base; //инициализация указателя на базовый класс p\_target полем p\_cl\_base  TYPE\_HANDLER p\_handler = connection -> p\_handler; //инициализация указателя p\_handler на метод TYPE\_HANDLER полем p\_handler  (p\_target ->\* p\_handler)(msg); //вызов метода p\_handler с передачей переменной msg по ссылке  }  }  }  string cl\_base::get\_way(){  string way = "/"; //инициализация строковой переменной way, равной "/"  cl\_base\* p = this; //инициализация указателя на базовый класс p равно this  if (!p -> get\_p\_head\_object()){return way;} //возвращенное значение метода get\_p\_head\_object() равно nullptr, возврат переменной way  way += p -> get\_s\_object\_name(); //добавление к переменной way возвращенного значения метода get\_s\_object\_name()  while (p -> get\_p\_head\_object() -> get\_p\_head\_object()){ //возвращенное значение метода get\_p\_head\_object() от возвращенного значения метода get\_p\_head\_object() равно nullptr  p = p -> get\_p\_head\_object(); //указатель р равен возвращенному значению метода get\_p\_head\_object() от указателя р  way = "/" + p -> get\_s\_object\_name() + way; //переменная way равно строке "/", возвращенному значению метода get\_s\_object\_name() и переменной way  }  return way; //возврат переменной way  }  void cl\_base::set\_ready\_for\_tree(int ready){  this -> ready = ready; //поле ready текущего объекта равно параметру ready  for (auto p\_sub : subordinate\_objects){p\_sub -> set\_ready\_for\_tree(ready);} //вызов метода set\_ready\_for\_tree с параметром ready для текущего объекта  } |

## Файл cl\_base.h

Листинг 64 – cl\_base.h

|  |
| --- |
| #ifndef CL\_BASE\_H  #define CL\_BASE\_H  #include <string>  #include <vector>  #include <iostream>  using namespace std;  #define SIGNAL\_D(signal\_f)(TYPE\_SIGNAL)(&signal\_f)  #define HANDLER\_D(handler\_f)(TYPE\_HANDLER)(&handler\_f)  class cl\_base;  typedef void(cl\_base::\*TYPE\_SIGNAL)(string&);  typedef void(cl\_base::\*TYPE\_HANDLER)(string&);  struct o\_sh{  TYPE\_SIGNAL p\_signal; //указатель на метод сигнала  cl\_base\* p\_cl\_base; //указатель на целевой объект  TYPE\_HANDLER p\_handler; //указатель на метод обработчика  };  class cl\_base{  protected:  string s\_object\_name; //наименование объекта  cl\_base \* p\_head\_object; //указатель на головной объект для текущего объекта  vector < cl\_base \* > subordinate\_objects; //массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии  int ready; //состояние готовности объекта  vector <o\_sh\*> connects; //вектор хранения установленных связей  public:  cl\_base(cl\_base\* p\_head\_object, string s\_object\_name = "Base\_object"); //параметризированный конструктор  bool set\_s\_object\_name(string s\_new\_name); //метод редактирования имени объекта  string get\_s\_object\_name(); //метод получения имени объекта  cl\_base \* get\_p\_head\_object(); //метод получения указателя на головной объект текущего объекта  cl\_base \* get\_subordinate\_objects(string s\_object\_name); //метод получения указателя на подчиненный объекта по его имени  ~cl\_base(); //для удаления дерева целиком (вызывает деструктор для всех объектов, подчиненных текущему)  cl\_base\* find\_object\_from\_root(string s\_name); //поиск объекта по имени от корневого объекта  cl\_base\*search\_object(string s\_name); //метод поиска объекта по имени от текущего объекта  int count(string s\_name); //метод определения количества вхождений объектов с данным именем от текущего объекта  cl\_base\*find\_object\_from\_current(string s\_name); //метод поиска объекта по имени от текущего объекта  void set\_ready(int i\_ready); //установкa готовности объекта  int get\_ready(); //получение состояние готовности объекта  cl\_base\* find\_root\_from\_current(); //метод поиска корневого объекта  cl\_base\* get\_point(string way); //метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты)  void print\_from\_current\_3(); //метод вывода иерархии объектов (дерева или ветки) от текущего объекта  int cl\_num; //номер класса объекта  void set\_connect(TYPE\_SIGNAL p\_signal, cl\_base\* p\_object, TYPE\_HANDLER p\_handler); //метод установления связи между сигналом текущего объекта и обработчиком целевого объекта  void delete\_connect(TYPE\_SIGNAL p\_signal, cl\_base\* p\_object, TYPE\_HANDLER p\_handler); //метод удаления связи между сигналом текущего объекта и обработчиком целевого объекта  void emit\_signal(TYPE\_SIGNAL p\_signal, string& msg); //метод выдачи сигнала от текущего объекта с передачей строковой переменной  string get\_way(); //метод получение абсолютного пути до текущего объекта  void set\_ready\_for\_tree(int ready); //метод установления состояния готовности для всех объектов  };  #endif |

## Файл main.cpp

Листинг 75 – main.cpp

|  |
| --- |
| #include "cl\_application.h"  int main(){  cl\_application ob\_cl\_application(nullptr); //создание объекта ob\_cl\_application класса cl\_application с помощью конструктора с параметром nullptr  ob\_cl\_application.build\_tree\_objects\_4(); //вызов метода build\_tree\_objects\_4() объекта ob\_cl\_application  return ob\_cl\_application.exec\_app\_4(); //вызов метода exec\_app\_4() объекта ob\_cl\_application  } |

# Приложение 2. Тестирование

Таблица 9 – Результат тестирования программы

| 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  EMIT /object\_s2/object\_s4 Send message 2  EMIT /object\_s2/object\_s4 Send message 3  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 to / Text: Send message 2 (class: 4)  Signal from /object\_s2/object\_s4  Signal to /object\_s2/object\_s6 Text: Send message 3 (class: 4)  Signal to / Text: Send message 3 (class: 4)  Signal from /object\_s1 | 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 to / Text: Send message 2 (class: 4)  Signal from /object\_s2/object\_s4  Signal to /object\_s2/object\_s6 Text: Send message 3 (class: 4)  Signal to / Text: Send message 3 (class: 4)  Signal from /object\_s1 |