

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc137229470)

[1 Постановка задачи 6](#_Toc137229471)

[2 Метод решения 11](#_Toc137229474)

[3 Описание алгоритмов 15](#_Toc137229475)

[4 Блок-схемы алгоритмов 26](#_Toc137229494)

[ЗАКЛЮЧЕНИЕ 39](#_Toc137229495)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc137229496)

[Приложение 1. Код программы 41](#_Toc137229497)

[Приложение 2. Тестирование 58](#_Toc137229513)

# введение

C++ – один из популярных языков программирования, поддерживающих ООП. Он имеет классы, объекты, наследование, необходимые для использования главных принципов ООП: инкапсуляция, наследование и полиморфизм. Всё это позволяет реализовать объектно-ориентированный подход, поэтому C++ часто используется для создания больших сложных программных систем.

Целью данной курсовой работы, является реализация механизма взаимодействия объектов с использованием сигналов и обработчиков, где на каждый сигнал свой обработчик. Сигнал должен передавать от источника к получателю, в процессе передачи сигналов будет обработка сигналов.

Для работы будет использоваться дерево иерархии объектов, которое будет строиться на основе введенных пользователем данных. Ввод иерархии будет осуществляться, как координаты объектов на дереве иерархии.

В рамках работы необходимо создать базовый класс, в котором будет реализован механизм установки и разрыва связей между сигналами и обработчиками, а также метод выдачи сигнала с передачей строковой переменной. Каждый класс будет содержать один метод сигнала и один метод обработчика.

Система будет запускаться и выполнять заданный пользователем набор команд, включая отправку сигналов, установку и удаление связей между объектами, а также изменение состояний объектов. Результаты работы системы будут выводиться в соответствии с описанным форматом.

Далее будет представлен метод решения, алгоритм и примеры входных и выходных данных для демонстрации работы сигналов и обработчиков с использованием множественного наследования классов на языке C++ [1, 2].

# 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:

1. Если текущий объект отключен, то выход, иначе к пункту 2.
2. Вызов метода сигнала с передачей строковой переменной по ссылке.
3. Цикл по всем связям сигнал-обработчик текущего объекта:
   1. 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.
4. 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 задачи необходимы те же классы, которые были использованы для решения работы КВ3, но с изменениями. Иерархия наследования классов представлена в Таблице 1.

**Класс cl\_base (с изменениями)**

**Добавлено:**

***Закрытое поле:***

* connections с типом vector<Connection\*> - хранит список соединений

***Открытые методы:***

* void emit(TYPE\_SIGNAL, string&) - выполняет обработку сигнала от текущего объекта
* void connect(TYPE\_SIGNAL, cl\_base\*, TYPE\_HANDLER) - устанавливает связь между сигналом текущего объекта и обработчиком целевого объекта
* void disconnect(TYPE\_SIGNAL, cl\_base\*, TYPE\_HANDLER) - разрывает связь между сигналом текущего объекта и обработчиком целевого объекта
* virtual void signal\_f(string&) - метод сигнала
* virtual void handler\_f(string) - метод обработчика

**Структура Connection**

**Добавлено:**

* TYPE\_SIGNAL p\_signal - указатель на метод сигнала
* cl\_base\* ob\_target - указатель на объект
* TYPE\_HANDLER p\_handler - указатель на метод обработчика

**Класс cl\_application (с изменениями)**

**Изменено:**

* void build\_tree\_objects() - строение иерархии объектов и устанавка связи
* int exec\_app() - обработка команды систем

**Класс cl\_2 (с изменениями)**

***Открытые методы:***

* void signal\_f(string&) - метод сигнала
* void handler\_f(string) - метод обработчика

**Класс cl\_3 (с изменениями)**

***Открытые методы:***

* void signal\_f(string&) - метод сигнала
* void handler\_f(string) - метод обработчика

**Класс cl\_4 (с изменениями)**

***Открытые методы:***

* void signal\_f(string&) - метод сигнала
* void handler\_f(string) - метод обработчика

**Класс cl\_5 (с изменениями)**

***Открытые методы:***

* void signal\_f(string&) - метод сигнала
* void handler\_f(string) - метод обработчика

**Класс cl\_6 (с изменениями)**

***Открытые методы:***

* void signal\_f(string&) - метод сигнала
* void handler\_f(string) - метод обработчика

*Таблица 1 – Иерархия наследования классов*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Имя класса | Классы-наследники | Моди-  фикатор доступа при насле-  довании | Описание | Номер | Ком-  мен-  тарий |
| 1 | cl\_base |  |  | Базовый класс, который содержит функциональность и свойства для построения иерархии объектов |  |  |
|  |  | cl\_application | public |  | 1 |  |
|  |  | cl\_2 | public |  | 2 |  |
|  |  | cl\_3 | public |  | 3 |  |
|  |  | cl\_4 | public |  | 4 |  |
|  |  | cl\_5 | public |  | 5 |  |
|  |  | cl\_6 | public |  | 6 |  |
| 2 | cl\_application |  |  | Класс корневого объекта иерархии объектов |  |  |

*Продолжение Таблицы 1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | cl\_2 |  |  | Класс объекта иерархии объектов |  |  |
| 4 | cl\_3 |  |  | Класс объекта иерархии объектов |  |  |
| 5 | cl\_4 |  |  | Класс объекта иерархии объектов |  |  |
| 6 | cl\_5 |  |  | Класс объекта иерархии объектов |  |  |
| 7 | cl\_6 |  |  | Класс объекта иерархии объектов |  |  |

# 3 ОПИСАНИЕ АЛГОРИТМОВ

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

## Алгоритм функции main

Функционал: главный метод программы.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | создание объекта ob\_cl\_application | 2 |
| 2 |  | вызов метода build\_tree\_objects() | 3 |
| 3 |  | вызов метода exec\_app() | ∅ |

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

Функционал: строение иерархии и установление связи.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода build\_tree\_objects класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | объявление переменных root\_name, child\_name строкового типа и class\_num целочисленного типа присвоение ей значения равно 0, ввод root\_name имени корневого объекта | 2 |
| 2 |  | установка имени корневого объекта | 3 |
| 3 |  | установка состояния корневого объекта в 1 | 4 |
| 4 |  | ввод root\_name координаты головного объекта | 5 |
| 5 | root\_name равно "endtree" |  | 9 |
|  | ввод child\_name имени дочернего объекта и class\_num номера класса | 6 |
| 6 |  | поиск головного объекта и номера класса | 7 |
| 7 | class\_num равно 2 | создание нового объекта класса cl\_2 с child\_name и p | 8 |
| class\_num равно 3 | создание нового объекта класса cl\_3 с child\_name и p | 8 |
| class\_num равно 4 | создание нового объекта класса cl\_4 с child\_name и p | 8 |
| class\_num равно 5 | создание нового объекта класса cl\_5 с child\_name и p | 8 |
| class\_num равно 6 | создание нового объекта класса cl\_6 с child\_name и p | 8 |
|  |  | 4 |
| 8 |  | установка состояния нового объекта в 1 | 4 |
| 9 |  | вызов метода print\_object\_tree() | 10 |
| 10 |  | ввод root\_name координаты объекта, выдающего сигнал | 11 |
| 11 | координат объекта равно "end\_of\_connections" |  | ∅ |
|  | ввод child\_name координаты целевого объекта | 12 |

*Продолжение Таблицы 3*

|  |  |  |  |
| --- | --- | --- | --- |
| 12 |  | поиск объекта, выдающего сигнал, по его координате | 13 |
| 13 |  | поиск целевого объекта по его координате | 14 |
| 14 | объект, выдающий сигнал, или целевой объект не найден |  | 10 |
|  | установление связи между сигналом объекта и обработчиком целевого объекта | 10 |

## Алгоритм метода exec\_app класса cl\_application

Функционал: обработка команд.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода exec\_app класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | объявление переменных cmd, path строкового типа и state целочисленного типа присвоение ей значения равно 0, ввод команды cmd | 2 |
| 2 | команда равно "END" |  | ∅ |
|  | ввод координаты path | 3 |
| 3 |  | поиск объекта по координате | 4 |
| 4 | объект не найден | вывод координата | 1 |
|  |  | 5 |
| 5 | команда равно "SET\_CONNECT" или команда равно "DELETE\_CONNECT" | вывод "Object координаты not found" | 6 |
|  |  | 8 |

*Продолжение Таблицы 4*

|  |  |  |  |
| --- | --- | --- | --- |
| 6 |  | поиск целевого объекта по координате | 7 |
| 7 | целевой объект не найден | вывод "Handler object координаты not found" | 1 |
|  |  | 8 |
| 8 | команда равно "EMIT" | ввод текста path | 9 |
|  | команда равно "SET\_CONDITION" | ввод состояния state | 10 |
| команда равно "SET\_CONNECT" | установка связи между объектом и целевым объектом | 1 |
| команда равно "DELETE\_CONNECT" | разрыв связи между объектом и целевым объектом | 1 |
|  |  | 1 |
| 9 |  | вызов метода emit объекта с передачей в него сигнала и текста | 1 |
| 10 |  | установка состояние объекта | 1 |

## Алгоритм метода emit класса cl\_base

Функционал: выдача сигнала от текущего объекта.

Параметры: TYPE\_SIGNAL, string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода emit класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | объект не готов |  | ∅ |
|  | вызов метода, на который указывает \*p\_signal от указателя на текущий объект, передав в него аргумент cmd | 2 |
| 2 | переменная-счетчик i равно 0 не выходит за пределы connections |  | 3 |
|  |  | ∅ |

*Продолжение Таблицы 5*

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | значения поля p\_signal элемента i равно значению аргумента p\_signal и значение вызова метода get\_state() от поля ob\_target элемента i | присвоение элементу i следующего элемента вектора connections  вызов метода, на который указывает \*p\_handler от указателя на объект ob\_target с аргументом cmd | 2 |
|  |  | 4 |
| 4 |  | увеличение переменной счетчика i на 1 | 2 |

## Алгоритм метода connect класса cl\_base

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

Параметры: TYPE\_SIGNAL, cl\_base\*, TYPE\_HANDLER.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода connect класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | переменная-счетчик i равно 0 не выходит за пределы connections |  | 2 |
|  |  | ∅ |
| 2 | поля p\_signal, ob\_target, p\_handler i-го элемента connections равны аргументам |  | ∅ |
|  |  | 3 |
|  |  | увеличение переменной-счетчика i на 1 | 1 |

## Алгоритм метода disconnect класса cl\_base

Функционал: разрывает связь между сигналом текущего объекта и обработчиком целевого объекта.

Параметры: TYPE\_SIGNAL, cl\_base\*, TYPE\_HANDLER.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода disconnect класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | it принадлежит connections |  | 2 |
|  |  | ∅ |
| 2 | значения, содержащиеся в структуре \*it, совпадают со значениями из параметров | удалить it из connections | ∅ |
|  |  | 1 |

## Алгоритм метода signal\_f класса cl\_base

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода signal\_f класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | вывод с новой строки "signal from", "абсолютного пути до текущего объекта" | ∅ |

## Алгоритм метода handler\_f класса cl\_base

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода handler\_f класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | вывод с новой строки "signal to", "абсолютный путь до текущего объекта" | ∅ |

## Алгоритм метода signal\_f класса cl\_2

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода signal\_f класса cl\_2

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | добавление к аргументу cmd строку "(class: 2)" | 2 |
| 2 |  | вывод с новой строки "signal from", "абсолютного пути до текущего объекта" | ∅ |

## Алгоритм метода handler\_f класса cl\_2

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода handler\_f класса cl\_2

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | вывод с новой строки "signal to", "абсолютный путь до текущего объекта" | ∅ |

## Алгоритм метода signal\_f класса cl\_3

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода signal\_f класса cl\_3

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | добавление к аргументу cmd строку "(class: 3)" | 2 |
| 2 |  | вывод с новой строки "signal from", "абсолютного пути до текущего объекта" | ∅ |

## Алгоритм метода handler\_f класса cl\_3

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода handler\_f класса cl\_3

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | вывод с новой строки "signal to", "абсолютный путь до текущего объекта" | ∅ |

## Алгоритм метода signal\_f класса cl\_4

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода signal\_f класса cl\_4

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | добавление к аргументу cmd строку "(class: 4)" | 2 |
| 2 |  | вывод с новой строки "signal from", "абсолютного пути до текущего объекта" | ∅ |

## Алгоритм метода handler\_f класса cl\_4

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода handler\_f класса cl\_4

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | вывод с новой строки "signal to", "абсолютный путь до текущего объекта" | ∅ |

## Алгоритм метода signal\_f класса cl\_5

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода signal\_f класса cl\_5

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | добавление к аргументу cmd строку "(class: 5)" | 2 |
| 2 |  | вывод с новой строки "signal from", "абсолютного пути до текущего объекта" | ∅ |

## Алгоритм метода handler\_f класса cl\_5

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода handler\_f класса cl\_5

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | вывод с новой строки "signal to", "абсолютный путь до текущего объекта" | ∅ |

## Алгоритм метода signal\_f класса cl\_6

Функционал: метод сигнала.

Параметры: string& cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 18.

Таблица 18 – Алгоритм метода signal\_f класса cl\_6

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | добавление к аргументу cmd строку "(class: 6)" | 2 |
| 2 |  | вывод с новой строки "signal from", "абсолютного пути до текущего объекта" | ∅ |

## Алгоритм метода handler\_f класса cl\_6

Функционал: метод обработчика.

Параметры: string cmd.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода handler\_f класса cl\_6

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | вывод с новой строки "signal to", "абсолютный путь до текущего объекта" | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

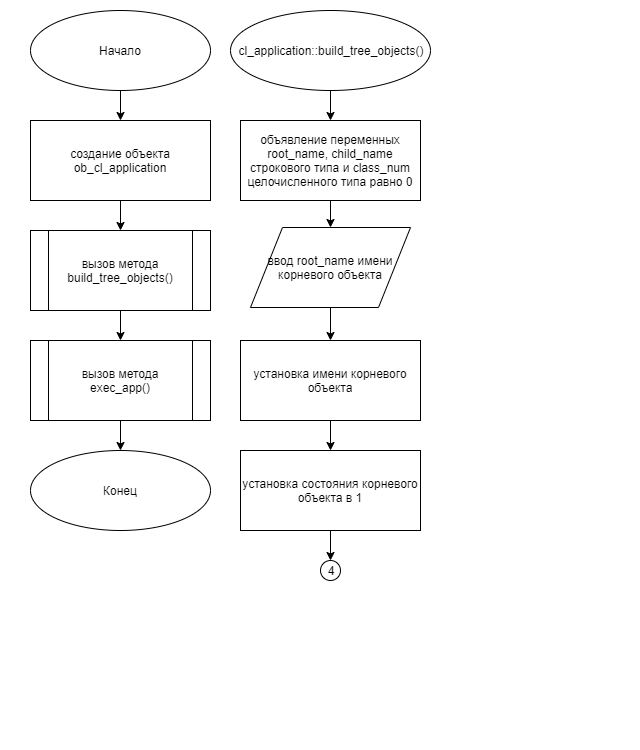


Рисунок 1 – Блок-схема алгоритма

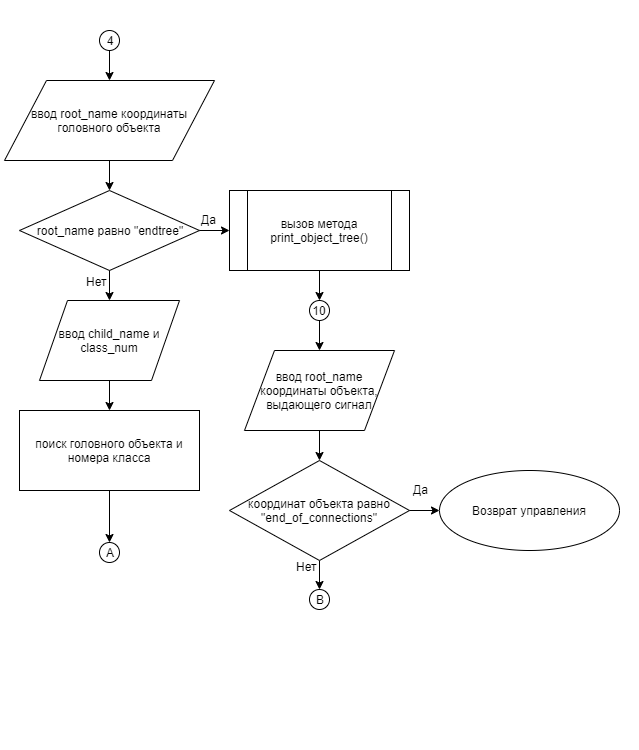


Рисунок 2 – Блок-схема алгоритма

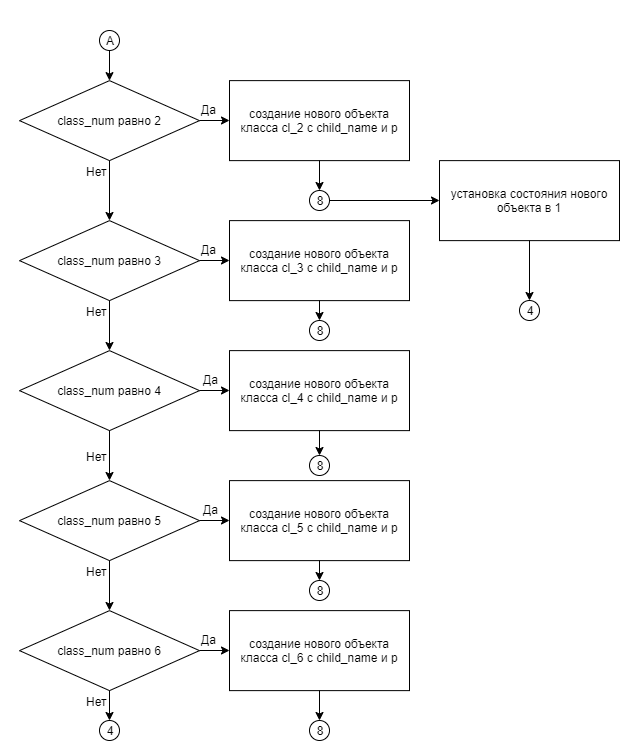


Рисунок 3 – Блок-схема алгоритма

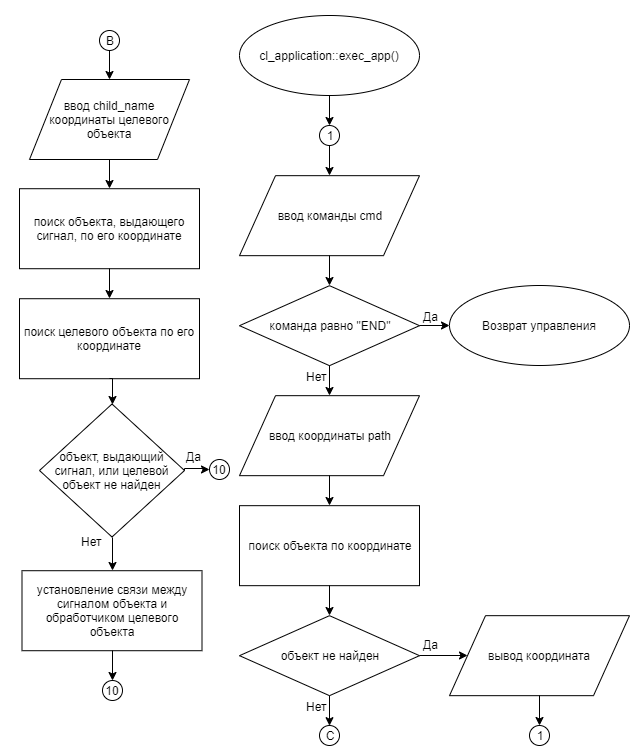


Рисунок 4 – Блок-схема алгоритма

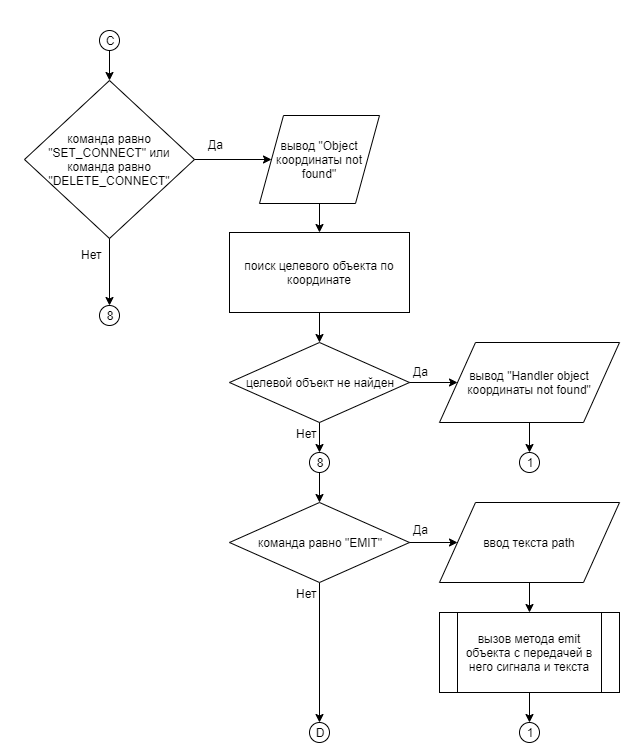


Рисунок 5 – Блок-схема алгоритма

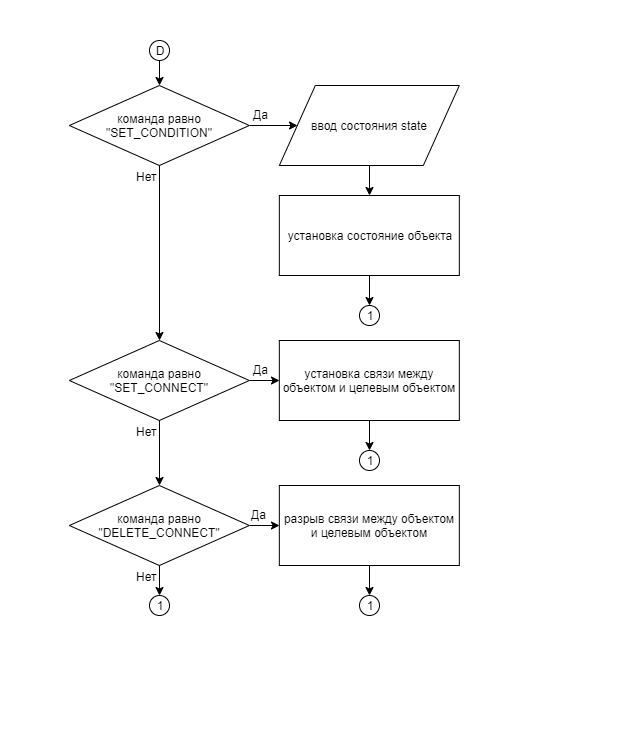


Рисунок 6 – Блок-схема алгоритма

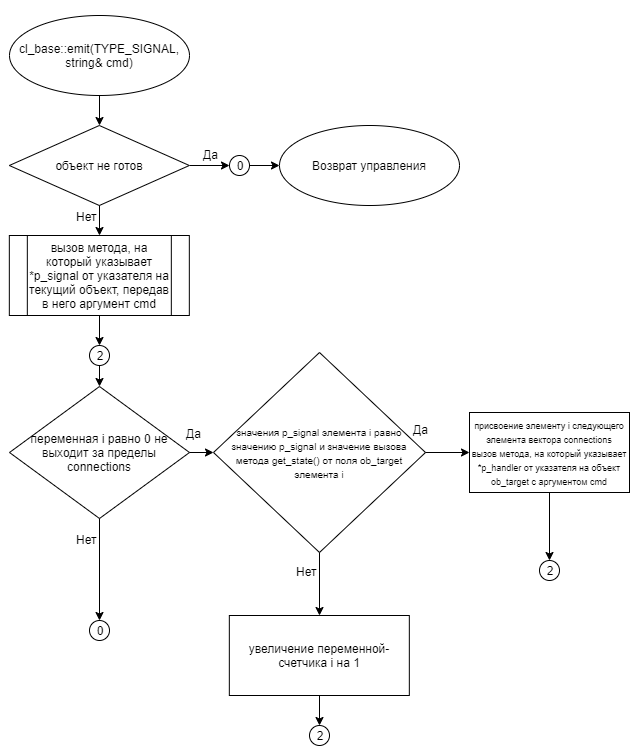


Рисунок 7 – Блок-схема алгоритма

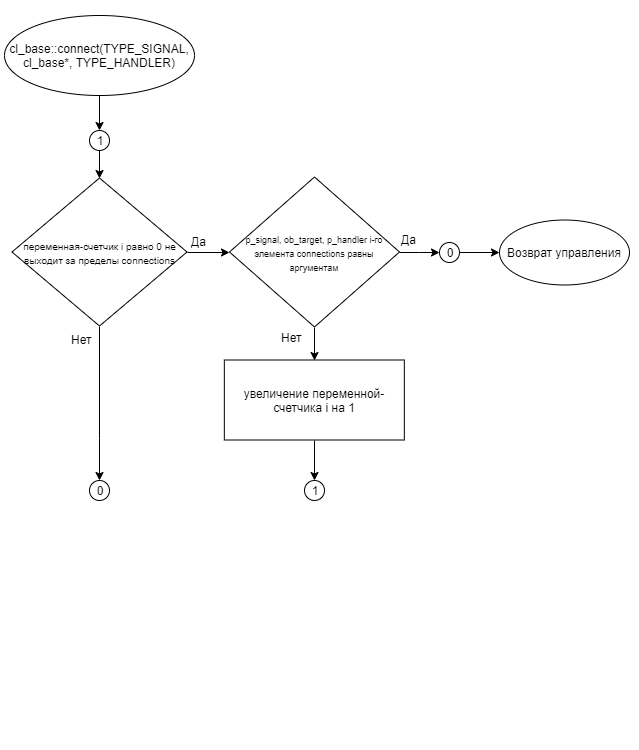


Рисунок 8 – Блок-схема алгоритма

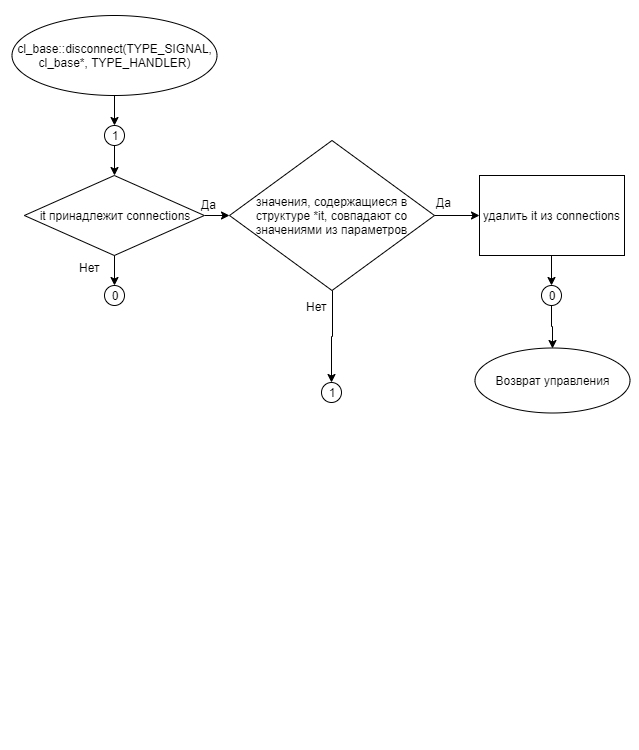


Рисунок 9 – Блок-схема алгоритма

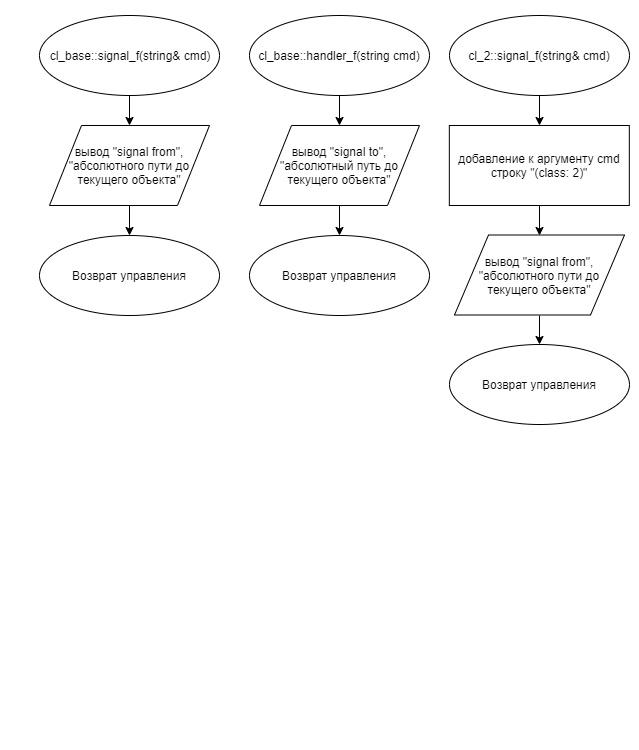


Рисунок 10 – Блок-схема алгоритма

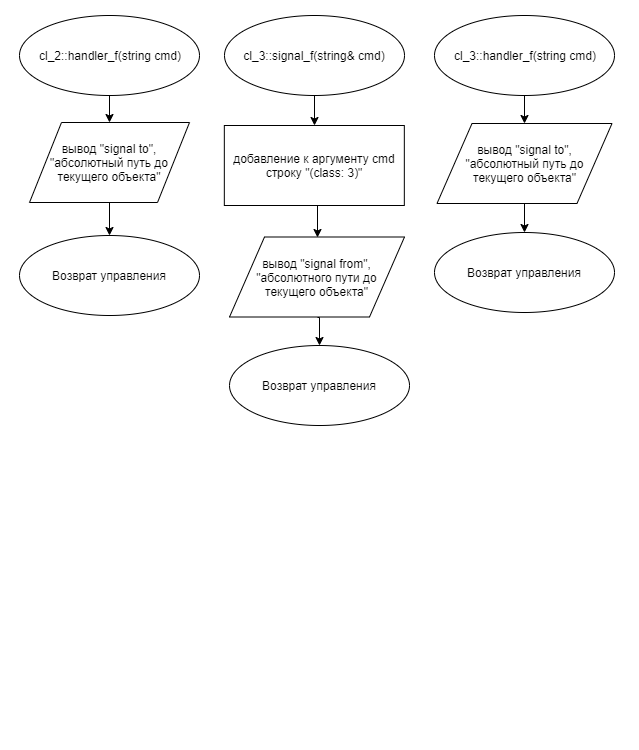


Рисунок 11 – Блок-схема алгоритма

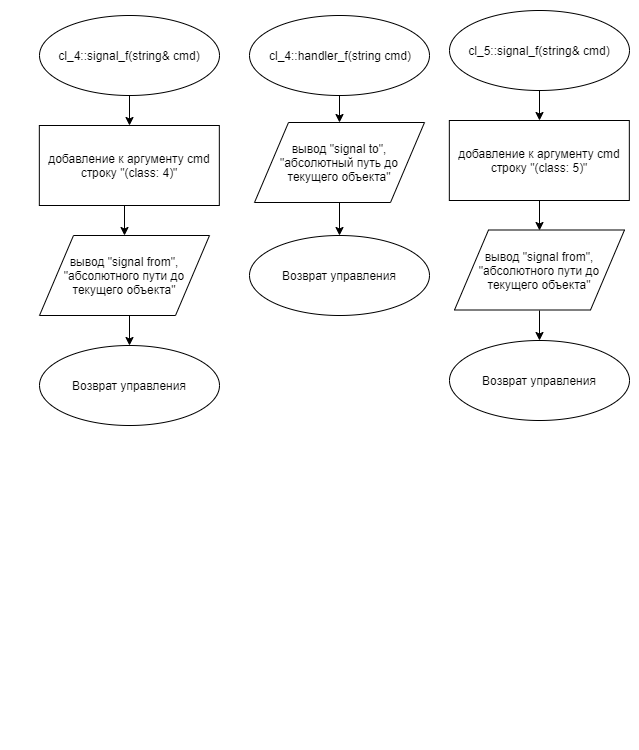


Рисунок 12 – Блок-схема алгоритма

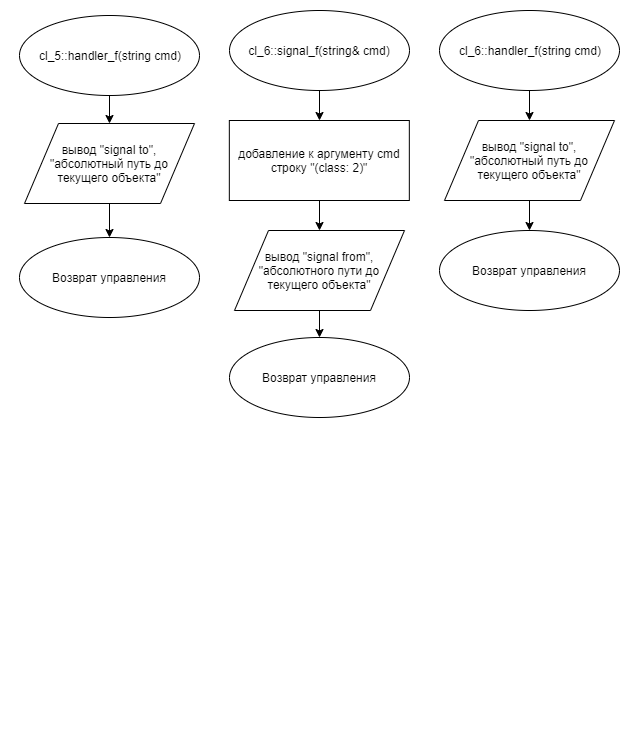


Рисунок 13 – Блок-схема алгоритма

# Заключение

В ходе выполнения курсовой работы, освоено ПО «Аврора» [5], было построено дерево иерархии объектов, было изменено и определено состояния объектов, было выведено на печать дерева иерархии объектов, был реализован механизма взаимодействия объектов с использованием сигналов и обработчиков, ознакомился с особенностями ООП на C++.

Таким образом, объектно-ориентированное программирование на C++ является удобным подходом к созданию сложных программных систем и важной дисциплиной для изучения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* + 1. ГОСТ 19 Единая система программной документации.
    2. Шилдт Г. С++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-

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

## Приложение 1. Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

## Файл cl\_2.cpp

Листинг 1 – cl\_2.cpp

|  |
| --- |
| #include "cl\_2.h"  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  cl\_2::cl\_2(cl\_base\* parent, string name) : cl\_base(parent, name) {};  void cl\_2::signal\_f(string& cmd)  {  cmd += " (class: 2)";  cout << endl << "Signal from " << this->get\_abs\_path();  }  void cl\_2::handler\_f(string cmd)  {  cout << endl << "Signal to " << this->get\_abs\_path() << " Text: " << cmd;  } |

## Файл cl\_2.h

Листинг 2 – cl\_2.h

|  |
| --- |
| #ifndef \_\_CL\_2\_\_H  #define \_\_CL\_2\_\_H  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class cl\_2 : public cl\_base  {  public: |

Продолжение Листинга 2

|  |
| --- |
| cl\_2(cl\_base\* parent=nullptr, string name="");  void signal\_f(string&);  void handler\_f(string);  };  #endif |

## Файл cl\_3.cpp

Листинг 3 – cl\_3.cpp

|  |
| --- |
| #include "cl\_3.h"  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  cl\_3::cl\_3(cl\_base\* parent, string name) : cl\_base(parent, name) {};  void cl\_3::signal\_f(string& cmd)  {  cmd += " (class: 3)";  cout << endl << "Signal from " << this->get\_abs\_path();  }  void cl\_3::handler\_f(string cmd)  {  cout << endl << "Signal to " << this->get\_abs\_path() << " Text: " << cmd;  } |

## Файл cl\_3.h

Листинг 4 – cl\_3.h

|  |
| --- |
| #ifndef \_\_CL\_3\_\_H  #define \_\_CL\_3\_\_H  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class cl\_3 : public cl\_base |

Продолжение Листинга 4

|  |
| --- |
| {  public:  cl\_3(cl\_base\* parent=nullptr, string name="");  void signal\_f(string&);  void handler\_f(string);  };  #endif |

## Файл cl\_4.cpp

Листинг 5 – cl\_4.cpp

|  |
| --- |
| #include "cl\_4.h"  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  cl\_4::cl\_4(cl\_base\* parent, string name) : cl\_base(parent, name) {};  void cl\_4::signal\_f(string& cmd)  {  cmd += " (class: 4)";  cout << endl << "Signal from " << this->get\_abs\_path();  }  void cl\_4::handler\_f(string cmd)  {  cout << endl << "Signal to " << this->get\_abs\_path() << " Text: " << cmd;  } |

## Файл cl\_4.h

Листинг 6 – cl\_4.h

|  |
| --- |
| #ifndef \_\_CL\_4\_\_H  #define \_\_CL\_4\_\_H  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector> |

Продолжение Листинга 6

|  |
| --- |
| using namespace std;  class cl\_4 : public cl\_base  {  public:  cl\_4(cl\_base\* parent=nullptr, string name="");  void signal\_f(string&);  void handler\_f(string);  };  #endif |

## Файл cl\_5.cpp

Листинг 7 – cl\_5.cpp

|  |
| --- |
| #include "cl\_5.h"  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  cl\_5::cl\_5(cl\_base\* parent, string name) : cl\_base(parent, name) {};  void cl\_5::signal\_f(string& cmd)  {  cmd += " (class: 2)";  cout << endl << "Signal from " << this->get\_abs\_path();  }  void cl\_5::handler\_f(string cmd)  {  cout << endl << "Signal to " << this->get\_abs\_path() << " Text: " << cmd;  } |

## Файл cl\_5.h

Листинг 8 – cl\_5.h

|  |
| --- |
| #ifndef \_\_CL\_5\_\_H  #define \_\_CL\_5\_\_H  #include "cl\_base.h" |

Продолжение Листинга 8

|  |
| --- |
| #include <iostream>  #include <string>  #include <vector>  using namespace std;  class cl\_5 : public cl\_base  {  public:  cl\_5(cl\_base\* parent=nullptr, string name="");  void signal\_f(string&);  void handler\_f(string);  };  #endif |

## Файл cl\_6.cpp

Листинг 9 – cl\_6.cpp

|  |
| --- |
| #include "cl\_6.h"  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  cl\_6::cl\_6(cl\_base\* parent, string name) : cl\_base(parent, name) {};  void cl\_6::signal\_f(string& cmd)  {  cmd += " (class: 6)";  cout << endl << "Signal from " << this->get\_abs\_path();  }  void cl\_6::handler\_f(string cmd)  {  cout << endl << "Signal to " << this->get\_abs\_path() << " Text: " << cmd;  } |

## Файл cl\_6.h

Листинг 10 – cl\_6.h

|  |
| --- |
| #ifndef \_\_CL\_6\_\_H  #define \_\_CL\_6\_\_H  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class cl\_6 : public cl\_base  {  public:  cl\_6(cl\_base\* parent=nullptr, string name="");  void signal\_f(string&);  void handler\_f(string);  };  #endif |

## Файл cl\_application.cpp

Листинг 11 – cl\_application.h

|  |
| --- |
| #include "cl\_application.h"  #include "cl\_2.h"  #include "cl\_3.h"  #include "cl\_4.h"  #include "cl\_5.h"  #include "cl\_6.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  cl\_application::cl\_application() : cl\_base(nullptr, "") {};  void cl\_application::build\_tree\_objects()  {  string root\_name, child\_name;  int class\_num = 0;  cin >> root\_name;  set\_object\_name(root\_name);  set\_state(1);  while (true)  {  cin >> root\_name;  if (root\_name == "endtree") |

Продолжение Листинга 11

|  |
| --- |
| {  break;  }  cin >> child\_name >> class\_num;  auto p = get\_by\_path(root\_name);  cl\_base\* obj;  switch (class\_num)  {  case 2:  obj = new cl\_2(p, child\_name);  break;  case 3:  obj = new cl\_3(p, child\_name);  break;  case 4:  obj = new cl\_4(p, child\_name);  break;  case 5:  obj = new cl\_5(p, child\_name);  break;  case 6:  obj = new cl\_6(p, child\_name);  break;  default:  obj = nullptr;  }  if (obj)  {  obj->set\_state(1);  }  }  print\_object\_tree();  while (true)  {  cin >> root\_name;  if (root\_name == "end\_of\_connections")  {  break;  }  cin >> child\_name;  auto found = get\_by\_path(root\_name);  auto target = get\_by\_path(child\_name);  if (!found || !target)  {  continue;  }  found->connect(SIGNAL\_D(cl\_base::signal\_f), target, HANDLER\_D(cl\_base::handler\_f));  }  } |

Продолжение Листинга 11

|  |
| --- |
| void cl\_application::print\_object\_tree()  {  cout << "Object tree" << endl;  show\_object\_tree(false);  }  int cl\_application::exec\_app()  {  string cmd, path;  int state = 0;  while (true)  {  cin >> cmd;  if (cmd == "END")  {  break;  }  cin >> path;  auto found = get\_by\_path(path);  if (!found)  {  cout << endl << "Object " << path << " not found";  continue;  }  cl\_base\* target{};  if (cmd == "SET\_CONNECT" || cmd == "DELETE\_CONNECT")  {  cin >> path;  target = get\_by\_path(path);  if (!target)  {  cout << endl << "Handler object " << path << " not found";  continue;  }  }  if (cmd == "EMIT")  {  getline(cin, path);  found->emit(SIGNAL\_D(cl\_base::signal\_f), path);  }  else if (cmd == "SET\_CONNECT")  {  found->connect(SIGNAL\_D(cl\_base::signal\_f), target, HANDLER\_D(cl\_base::handler\_f));  }  else if (cmd == "DELETE\_CONNECT")  {  found->disconnect(SIGNAL\_D(cl\_base::signal\_f), target, HANDLER\_D(cl\_base::handler\_f));  }  else if (cmd == "SET\_CONDITION")  {  cin >> state;  found->set\_state(state);  }  } |

Продолжение Листинга 11

|  |
| --- |
| return 0;  } |

## Файл cl\_application.h

Листинг 12 – cl\_application.h

|  |
| --- |
| #ifndef \_\_CL\_APPLICATION\_\_H  #define \_\_CL\_APPLICATION\_\_H  #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class cl\_application : public cl\_base  {  public:  cl\_application();  void build\_tree\_objects();  void print\_object\_tree();  int exec\_app();  };  #endif |

## Файл cl\_base.cpp

Листинг 13 – cl\_base.cpp

|  |
| --- |
| #include "cl\_base.h"  #include <iostream>  #include <string>  #include <vector>  using namespace std;  cl\_base::cl\_base(cl\_base\* parent, string name): p\_parent(parent), object\_name(name)  {  if (parent != nullptr)  {  parent->children.push\_back(this);  }  } |

Продолжение Листинга 13

|  |
| --- |
| cl\_base::~cl\_base()  {  for (auto c: children)  {  delete c;  }  }  bool cl\_base::set\_object\_name(string object\_name)  {  if (!p\_parent)  {  this->object\_name = object\_name;  return true;  }  it\_child = children.begin();  while (it\_child != children.end())  {  if (((\*it\_child)->get\_object\_name() == object\_name) && ((\*it\_child) != this))  {  return false;  }  it\_child++;  }  this->object\_name = object\_name;  return true;  }  string cl\_base::get\_object\_name()  {  return object\_name;  }  cl\_base\* cl\_base::get\_parent()  {  return p\_parent;  }  void cl\_base::show\_object\_tree(bool show\_state)  {  show\_object\_next(0, show\_state);  }  void cl\_base::show\_object\_next(int i\_level, bool show\_state)  {  string s\_space = "";  if (i\_level > 0)  {  s\_space.append(4 \* i\_level, ' ');  }  cout << s\_space << get\_object\_name();  if (show\_state)  { |

Продолжение Листинга 13

|  |
| --- |
| if (get\_state())  {  cout << " is ready";  }  else  {  cout << " is not ready";  }  }  for (auto c : children)  {  cout << endl;  c->show\_object\_next(i\_level + 1, show\_state);  }  }  cl\_base\* cl\_base::get\_child(string child\_name)  {  for (auto child : children)  {  if (child->get\_object\_name() == child\_name)  {  return child;  }  }  return nullptr;  }  bool cl\_base::get\_state()  {  return state;  }  void cl\_base::set\_state(bool state)  {  if (get\_parent() && !get\_parent()->get\_state())  {  this->state = false;  }  else  {  this->state = state;  }  if (!state)  {  for (auto c : children)  {  c->set\_state(state);  }  }  }  cl\_base\* cl\_base::find(string object\_name)  {  int count = 0;  auto found = find\_internal(object\_name, count); |

Продолжение Листинга 13

|  |
| --- |
| if (count > 1)  {  return nullptr;  }  return found;  }  cl\_base\* cl\_base::find\_internal(string object\_name, int& count)  {  cl\_base\* found = nullptr;  if (object\_name == get\_object\_name())  {  count++;  found = this;  }  for (auto c : children)  {  auto obj = c->find\_internal(object\_name, count);  if (obj)  {  found = obj;  }  }  return found;  }  cl\_base\* cl\_base::find\_on\_whole\_tree(string object\_name)  {  auto p = this;  while (p->get\_parent())  {  p = p->get\_parent();  }  return p->find(object\_name);  }  bool cl\_base::is\_subordinate(cl\_base\* obj)  {  for (auto child: children)  {  if (child == obj)  {  return true;  }  if (child->is\_subordinate(obj))  {  return true;  }  }  return false;  }  bool cl\_base::move(cl\_base\* new\_parent)  {  if (!get\_parent() || !new\_parent || new\_parent- |

Продолжение Листинга 13

|  |
| --- |
| >get\_child(get\_object\_name()) || is\_subordinate(new\_parent))  {  return false;  }  get\_parent()->remove\_child(this);  new\_parent->children.push\_back(this);  p\_parent = new\_parent;  return true;  }  void cl\_base::delete\_child(string child\_name)  {  for (auto it = children.begin(); it != children.end(); it++)  {  if ((\*it)->get\_object\_name() == child\_name)  {  delete \*it;  children.erase(it);  break;  }  }  }  cl\_base\* cl\_base::get\_by\_path(string path)  {  if (path.substr(0, 2) == "//")  {  return find\_on\_whole\_tree(path.substr(2));  }  else if (path == ".")  {  return this;  }  else if (path[0] == '.')  {  return find(path.substr(1));  }  else if (path[0] == '/')  {  auto new\_path = path.substr(1);  auto cur = this;  while (cur->get\_parent())  {  cur = cur->get\_parent();  }  if (path == "/")  {  return cur;  }  return cur->get\_by\_path(new\_path);  }  auto pos = path.find('/');  auto child = get\_child(path.substr(0, pos));  if (!child || pos == string::npos) // -1  {  return child;  }  return child->get\_by\_path(path.substr(pos + 1));  } |

Продолжение Листинга 13

|  |
| --- |
| void cl\_base::remove\_child(cl\_base\* child)  {  for (auto it = children.begin(); it != children.end(); it++)  {  if ((\*it) == child)  {  children.erase(it);  break;  }  }  }  string cl\_base::get\_abs\_path()  {  if (!get\_parent())  {  return "/";  }  auto path = get\_parent()->get\_abs\_path() + "/" + get\_object\_name();  if (path.substr(0, 2) == "//")  {  return path.substr(1);  }  return path;  }  void cl\_base::emit(TYPE\_SIGNAL p\_signal, string& cmd)    {  TYPE\_HANDLER p\_handler;  cl\_base\* ob\_target;  if (!get\_state())  {  return;  }  (this->\*p\_signal) (cmd);  for (unsigned int i = 0; i < connections.size(); i++)  {  if (connections[i]->p\_signal == p\_signal && connections[i]->ob\_target->get\_state())  {  p\_handler = connections[i]->p\_handler;  ob\_target = connections[i]->ob\_target;  (ob\_target->\*p\_handler) (cmd);  }  }  }  void cl\_base::connect(TYPE\_SIGNAL p\_signal, cl\_base\* ob\_target, TYPE\_HANDLER p\_handler)  {  for (unsigned int i = 0; i < connections.size(); i++)  {  if (connections[i]->p\_signal == p\_signal && connections[i]->ob\_target == ob\_target && connections[i]->p\_handler == p\_handler)  {  return; |

Продолжение Листинга 13

|  |
| --- |
| }  }  Connection\* sh\_temp = new Connection;  sh\_temp->p\_signal = p\_signal;  sh\_temp->ob\_target = ob\_target;  sh\_temp->p\_handler = p\_handler;  connections.push\_back(sh\_temp);  }  void cl\_base::disconnect(TYPE\_SIGNAL p\_signal, cl\_base\* ob\_target, TYPE\_HANDLER p\_handler)  {  for (auto it = connections.begin(); it != connections.end(); it++)  {  if ((\*it)->p\_signal == p\_signal && (\*it)->ob\_target == ob\_target && (\*it)->p\_handler == p\_handler)  {  connections.erase(it);  break;  }  }  }  void cl\_base::signal\_f(string& cmd)  {  cout << endl << "Signal from " << this->get\_abs\_path();  }  void cl\_base::handler\_f(string cmd)  {  cout << endl << "Signal to " << this->get\_abs\_path() << " Text: " << cmd;  } |

## Файл cl\_base.h

Листинг 14 – cl\_base.h

|  |
| --- |
| #ifndef \_\_CL\_BASE\_\_H  #define \_\_CL\_BASE\_\_H  #include <iostream>  #include <string>  #include <vector>  #include <typeinfo>  using namespace std;  class cl\_base;  typedef void (cl\_base::\* TYPE\_SIGNAL) (string&);  typedef void (cl\_base::\* TYPE\_HANDLER) (string); |

Продолжение Листинга 14

|  |
| --- |
| #define SIGNAL\_D( signal\_f ) ( TYPE\_SIGNAL ) ( & signal\_f )  #define HANDLER\_D( handler\_f ) (TYPE\_HANDLER ) ( & handler\_f )  struct Connection  {  TYPE\_SIGNAL p\_signal;  cl\_base\* ob\_target;  TYPE\_HANDLER p\_handler;  };  class cl\_base  {  private:  string object\_name;  cl\_base\* p\_parent;  bool state;  vector <Connection\*> connections;  public:  vector <cl\_base\*> children;  vector <cl\_base\*> :: iterator it\_child;  cl\_base(cl\_base\* parent=nullptr, string name="");  ~cl\_base();  bool set\_object\_name(string object\_name);  string get\_object\_name();  cl\_base\* get\_parent();  void show\_object\_tree(bool show\_state=false);  void show\_object\_next(int i\_level, bool show\_state);  cl\_base\* get\_child(string child\_name);  bool get\_state();  void set\_state(bool state);  cl\_base\* find(string object\_name);  cl\_base\* find\_on\_whole\_tree(string object\_name);  void remove\_child(cl\_base\* child);  cl\_base\* find\_internal(string object\_name, int& count);  bool move(cl\_base\* new\_parent);  void delete\_child(string child\_name);  cl\_base\* get\_by\_path(string path);  string get\_abs\_path();  bool is\_subordinate(cl\_base\* obj);  void emit(TYPE\_SIGNAL p\_signal, string& cmd);  void connect(TYPE\_SIGNAL p\_signal, cl\_base\* ob\_target, TYPE\_HANDLER p\_handler);  void disconnect(TYPE\_SIGNAL p\_signal, cl\_base\* ob\_target, TYPE\_HANDLER p\_handler);  virtual void signal\_f(string& cmd);  virtual void handler\_f(string cmd);  };  #endif |

## Файл main.cpp

Листинг 15 – main.cpp

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <iostream>  #include "cl\_application.h"  using namespace std;  int main()  {  cl\_application ob\_cl\_application;  ob\_cl\_application.build\_tree\_objects();  ob\_cl\_application.exec\_app();  return(0);  } |

## Приложение 2. Тестирование

Результат тестирования программы представлен в Таблице 20.

Таблица 20 – 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 |