#### **СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc137574763)

[1 ПОСТАНОВКА ЗАДАЧИ 8](#_Toc137574764)

[1.1 Описание входных данных 10](#_Toc137574765)

[1.2 Описание выходных данных 12](#_Toc137574766)

[2 МЕТОД РЕШЕНИЯ 14](#_Toc137574767)

[3 ОПИСАНИЕ АЛГОРИТМОВ 16](#_Toc137574768)

[3.1 Алгоритм метода set\_connection класса cl\_base 16](#_Toc137574769)

[3.2 Алгоритм метода delete\_connection класса cl\_base 17](#_Toc137574770)

[3.3 Алгоритм метода emit\_signal класса cl\_base 17](#_Toc137574771)

[3.4 Алгоритм метода setConnections класса cl\_application 18](#_Toc137574772)

[3.5 Алгоритм конструктора класса cl\_1 20](#_Toc137574773)

[3.6 Алгоритм конструктора класса cl\_2 20](#_Toc137574774)

[3.7 Алгоритм конструктора класса cl\_3 20](#_Toc137574775)

[3.8 Алгоритм конструктора класса cl\_4 21](#_Toc137574776)

[3.9 Алгоритм конструктора класса cl\_6 21](#_Toc137574777)

[3.10 Алгоритм конструктора класса cl\_5 21](#_Toc137574778)

[3.11 Алгоритм метода signal класса cl\_1 22](#_Toc137574779)

[3.12 Алгоритм метода signal класса cl\_2 22](#_Toc137574780)

[3.13 Алгоритм метода signal класса cl\_3 22](#_Toc137574781)

[3.14 Алгоритм метода signal класса cl\_4 23](#_Toc137574782)

[3.15 Алгоритм метода signal класса cl\_5 23](#_Toc137574783)

[3.16 Алгоритм метода signal класса cl\_6 24](#_Toc137574784)

[3.17 Алгоритм метода handler класса cl\_1 24](#_Toc137574785)

[3.18 Алгоритм метода handler класса cl\_2 24](#_Toc137574786)

[3.19 Алгоритм метода handler класса cl\_3 25](#_Toc137574787)

[3.20 Алгоритм метода handler класса cl\_4 25](#_Toc137574788)

[3.21 Алгоритм метода handler класса cl\_5 26](#_Toc137574789)

[3.22 Алгоритм метода handler класса cl\_6 26](#_Toc137574790)

[3.23 Алгоритм функции main 26](#_Toc137574791)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 28](#_Toc137574792)

[5 КОД ПРОГРАММЫ 41](#_Toc137574793)

[5.1 Файл cl\_1.cpp 41](#_Toc137574794)

[5.2 Файл cl\_1.h 41](#_Toc137574795)

[5.3 Файл cl\_2.cpp 42](#_Toc137574796)

[5.4 Файл cl\_2.h 42](#_Toc137574797)

[5.5 Файл cl\_3.cpp 43](#_Toc137574798)

[5.6 Файл cl\_3.h 43](#_Toc137574799)

[5.7 Файл cl\_4.cpp 44](#_Toc137574800)

[5.8 Файл cl\_4.h 44](#_Toc137574801)

[5.9 Файл cl\_5.cpp 45](#_Toc137574802)

[5.10 Файл cl\_5.h 45](#_Toc137574803)

[5.11 Файл cl\_6.cpp 46](#_Toc137574804)

[5.12 Файл cl\_6.h 46](#_Toc137574805)

[5.13 Файл cl\_application.cpp 47](#_Toc137574806)

[5.14 Файл cl\_application.h 50](#_Toc137574807)

[5.15 Файл cl\_base.cpp 51](#_Toc137574808)

[5.16 Файл cl\_base.h 58](#_Toc137574809)

[5.17 Файл main.cpp 59](#_Toc137574810)

[6 ТЕСТИРОВАНИЕ 60](#_Toc137574811)

[ЗАКЛЮЧЕНИЕ 62](#_Toc137574812)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 63](#_Toc137574813)

# ВВЕДЕНИЕ

Язык программирования C++ - компилируемый и универсальный язык общего назначения, наследник языка C. Он сочетает в себе низкоуровневые функции C с высокоуровневыми абстракциями и возможностями объектно-ориентированного программирования - современной парадигмы разработки, целью которой является структурирование кода на основе концепции объектов, являющихся экземплярами классов.

В рамках изучения курса объектно-ориентированного программирования требовалось выполнение курсовой работы на основе задания из системы "Аврора" по выбранной теме с целью освоения программы по вышеназванному предмету, получению навыков разработки программ на языке C++ и решению задач, содержащихся непосредственно в самой формулировке задания.

Целью курсовой работы является моделирование работы башенного крана, используя все возможности объектно-ориентированного программирования с использованием сигналов и обработчиков.

В связи с поставленной целью необходимо выполнить следующие задачи:

* Освоение процедурного программирования на языке C++;
* Освоение объектно-ориентированного языка программирования С++;
* Освоение умения разработки программы как системы;
* Умение использовать отладку программы;
* Освоение работы с многофайловыми программами;
* Освоение умения проектирования архитектуры программы на базе построения иерархии объектов;
* Освоение версионности при разработке программ;
* Освоение выполнения всех необходимых работ согласно этапам разработки;
* программы и соответствующих программных инструментов;
* Моделирование работы башенного крана при помощи сигналов и обработчиков;
* Построение системы взаимодействия объектов посредством интерфейса сигналов и обработчиков;
* Описание алгоритма работы программы;
* Построение блок схемы алгоритма работы программы;
* Написание кода на языке программирования С++, согласно разработанному алгоритму работы программы;

# 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 синтаксический корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы. Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.

## Описание входных данных

#### В методе построения системы.

#### Множество объектов, их характеристики и расположение на дереве

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

#### После ввода состава дерева иерархии построчно вводится:

«координата объекта выдающего сигнал» «координата целевого объекта»

#### Ввод информации для построения связей завершается строкой, которая содержит:

«end\_of\_connections»

#### В методе запуска (отработки) системы построчно вводятся множество команд в производном порядке:

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

#### SET\_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – установка связи;

#### DELETE\_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаление связи;

#### SET\_CONDITION «координата объекта» «значение состояния» – установка состояния объекта.

#### END – завершить функционирование системы (выполнение программы). Команда END присутствует обязательно.

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

#### Если не найден объект по координате:

Object «координата объекта» not found

#### Если не найден целевой объект по координате:

Handler object «координата целевого объекта» not found

### Пример ввода:

appls\_root

/ object\_s1 3

/ object\_s2 2

/object\_s2 object\_s4 4

/ object\_s13 5

/object\_s2 object\_s6 6

/object\_s1 object\_s7 2 endtree

/object\_s2/object\_s4 /object\_s2/object\_s6

/object\_s2 /object\_s1/object\_s7

/ /object\_s2/object\_s4

/object\_s2/object\_s4 / end\_of\_connections

EMIT /object\_s2/object\_s4 Send message 1 EMIT /object\_s2/object\_s4 Send message 2 EMIT /object\_s2/object\_s4 Send message 3 EMIT /object\_s1 Send message 4

END

## Описание выходных данных

#### Первая строка:

Object tree

#### Со второй строки вывести иерархию построенного дерева. Далее, построчно, если отработал метод сигнала:

Signal from «абсолютная координата объекта»

#### Если отработал метод обработчика:

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

### Пример вывода:

Object tree appls\_root

object\_s1

object\_s7 object\_s2

object\_s4 object\_s6

object\_s13

Signal from /object\_s2/object\_s4

Signal to /object\_s2/object\_s6 Text: Send message 1 (class: 4) Signal to / Text: Send message 1 (class: 4)

Signal from /object\_s2/object\_s4

Signal to /object\_s2/object\_s6 Text: Send message 2 (class: 4) Signal 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

# МЕТОД РЕШЕНИЯ

#### Для решения задачи были изменены:

#### Класс cl\_1

#### Структура oSh

#### Пользовательские типы данных TYPE\_SIGNAL, TYPE\_HANDLER

#### Параметризированное макроопределение препроцессора

#### Класс cl\_base

#### Поля

####  Целочисленное поле number

####  Тип данных - целый тип

####  Название - number

####  Модификатор доступа - public

####  Ветор connects структур oSh

####  Тип данных - вектор структур oSh

####  Название - connects

####  Модификатор доступа - public

#### Методы

####  Метод setСonnect

####  Функционал - устанавливает связь между двумя объектами

####  Метод deleteСonnect

####  Функционал - удаляет связь между двумя объектами

####  Метод emitSignal

####  Функционал-отправляетсообщениеотодногообъектак другому

####  Метод setFullReadiness

####  Функционал - устанваливает готовность на все деревья в дереве

#### Класс cl\_application

#### Методы

####  Метод setConnections

####  Функционал - обработка установления связи между объектами

####  Метод handleCommands

####  Функционал - обработка ввода комманд

#### Классы cl\_1 по класс cl\_6:

#### Методы

####  Название - pSignal

####  Поле pClObject

####  Тип данных - указатель на объект класса cl\_base

####  Модификатор - public

####  Название - pClObject

####  Поле pHandler

####  Тип данных - пользовательский тип TYPE\_HANDLER

####  Модификатор - public

####  Название - pHandler

# ОПИСАНИЕ АЛГОРИТМОВ

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

## Алгоритм метода set\_connection класса cl\_base

#### Функционал: Устанавливает связь между двумя объектами.

#### Параметры: пользовательский тип TYPE\_SIGNAL, указатель на объект класса cl\_base, пользовательский тип TYPE\_HANDLER.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 1.

###### Таблица 1 – Алгоритм метода set\_connection класса cl\_base

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Инициализация pValue указателя на структуру  oSh | 2 |
| 2 | i меньше длины вектора  connects |  | 3 |
|  |  | 4 |
| 3 | Все поля текущей связи совпадают с переданными  параметрами |  | ∅ |
|  |  | 2 |
| 4 |  | Присвоение pValue значения указателя на  структуру oSh | 5 |
| 5 |  | Присвоение полю p\_signal значения параметра  p\_signal | 6 |
| 6 |  | Присвоение полю p\_handler значения параметра | 7 |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
|  |  | p\_handler |  |
| 7 |  | Присвоение полю p\_target значения параметра  p\_target | 8 |
| 8 |  | Добавление в вектор connects переменной p\_value | ∅ |

## Алгоритм метода delete\_connection класса cl\_base

#### Функционал: Удаляет связь между двумя объектами.

#### Параметры: пользовательский тип TYPE\_SIGNAL, указатель на объект класса cl\_base, пользовательский тип TYPE\_HANDLER.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 2.

###### Таблица 2 – Алгоритм метода delete\_connection класса cl\_base

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | создание итератора p\_it | 2 |
| 2 | i меньше длины вектора  connects |  | 3 |
|  |  | ∅ |
| 3 | Все поля текущей связи  совпадают с переданными параметрами | Удаление из списка connects текущей связи | 2 |
|  |  | 2 |

## Алгоритм метода emit\_signal класса cl\_base

#### Функционал: Отправка сообщения от одного объекта к другому. Параметры: пользовательский тип TYPE\_SIGNAL, адрес строкового типа.

#### Возвращаемое значение: отсутствует.

#### Алгоритм метода представлен в таблице 3.

###### Таблица 3 – Алгоритм метода emit\_signal класса cl\_base

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | вызываем указанный сигнал, передавая ему  сообщение s\_massege. | 1 |
| 2 | начинаем перебор всех соединений, которые установлены для данного  объекта |  | 2 |
|  |  | 3 |
| 3 | проверяем, соответствует ли текущее соединение  вызванному сигналу. |  | 4 |
|  |  | ∅ |
| 4 |  | сохраняем указатель на объект-получатель,  который был связан с данным соединением | 5 |
| 5 |  | сохраняем указатель на метод-обработчик, который  был связан с данным соединением | 6 |
| 6 |  | вызываем метод-обработчик на объекте-  получателе, передавая ему сообщение s\_massege | ∅ |

## Алгоритм метода setConnections класса cl\_application

#### Функционал: Управляет установкой связью между объектов. Параметры: отсутствуют.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 4.

###### Таблица 4 – Алгоритм метода setConnections класса cl\_application

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Инициализация строковой переменной  sender\_coordinate | 2 |
| 2 |  | Инициализация строковой переменной reciever-  coordinate | 3 |
| 3 |  | Инициализация указателя на объект класса Base  p\_sender | 4 |
| 4 |  | Инициализация указателя на объект класса Base  p\_reciever | 5 |
| 5 |  | Создание списка методов сигнала всех классов | 6 |
| 6 |  | Создание списка обработчиков сигнала всех  классов | 7 |
| 7 |  | Ввод переменной sender\_coordinate | 8 |
| 8 | sender-coordinate равно  "end\_of\_connections" |  | ∅ |
|  |  | 9 |
| 9 |  | Ввод переменной reciever\_coordinate | 10 |
| 1  0 |  | Присвоение p\_sender значения вызова методa  find\_obj\_by\_coord | 11 |
| 1  1 |  | Присвоение p\_reciever значения вызова метода  find\_obj\_by\_coord | 12 |
| 1  2 |  | Присвоение переменной signal значения элемента списка SIGNALS\_LIST под индексом текущего  класса минус 1 | 13 |
| 1  3 |  | Присвоение переменной handler значения элемента списка HANDLERS\_LIST под индексом текущего  класса минус 1 | 14 |
| 1  4 |  | Вызов метода set\_connection от p\_sender | 7 |

## Алгоритм конструктора класса cl\_1

#### Функционал: Параметризированный конструктор объекта. Параметры: указатель на объект класса Base, строка.

#### Алгоритм конструктора представлен в таблице 5.

###### Таблица 5 – Алгоритм конструктора класса cl\_1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Присвоение полю cl\_number номера текущего класса | ∅ |

## Алгоритм конструктора класса cl\_2

#### Функционал: Параметризированный конструктор объекта. Параметры: указатель на объект класса Base, строка.

#### Алгоритм конструктора представлен в таблице 6.

###### Таблица 6 – Алгоритм конструктора класса cl\_2

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Присвоение полю cl\_number номера текущего класса | ∅ |

## Алгоритм конструктора класса cl\_3

#### Функционал: Параметризированный конструктор объекта. Параметры: указатель на объект класса Base, строка.

#### Алгоритм конструктора представлен в таблице 7.

###### Таблица 7 – Алгоритм конструктора класса cl\_3

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Присвоение полю cl\_number номера текущего класса | ∅ |

## Алгоритм конструктора класса cl\_4

#### Функционал: Параметризированный конструктор объекта. Параметры: указатель на объект класса Base, строка.

#### Алгоритм конструктора представлен в таблице 8.

###### Таблица 8 – Алгоритм конструктора класса cl\_4

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Присвоение полю cl\_number номера текущего класса | ∅ |

## Алгоритм конструктора класса cl\_6

#### Функционал: Параметризированный конструктор объекта. Параметры: указатель на объект класса Base, строка.

#### Алгоритм конструктора представлен в таблице 9.

###### Таблица 9 – Алгоритм конструктора класса cl\_6

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Присвоение полю cl\_number номера текущего класса | ∅ |

## Алгоритм конструктора класса cl\_5

#### Функционал: Параметризированный конструктор объекта. Параметры: указатель на объект класса Base, строка.

#### Алгоритм конструктора представлен в таблице 10.

###### Таблица 10 – Алгоритм конструктора класса cl\_5

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Присвоение полю cl\_number номера текущего класса | ∅ |

## Алгоритм метода signal класса cl\_1

#### Функционал: Метод сигнала.

#### Параметры: Адрес строковой переменной. Возвращаемое значение: Отсутствует.

#### Алгоритм метода представлен в таблице 11.

###### Таблица 11 – Алгоритм метода signal класса cl\_1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal from" координата текущего объекта | 2 |
| 2 |  | Добавление к строке текста "class: " номер класса текущего объекта | ∅ |

## Алгоритм метода signal класса cl\_2

#### Функционал: Метод сигнала.

#### Параметры: Адрес строковой переменной. Возвращаемое значение: Отсутствует.

#### Алгоритм метода представлен в таблице 12.

###### Таблица 12 – Алгоритм метода signal класса cl\_2

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal from" координата текущего объекта | 2 |
| 2 |  | Добавление к строке текста "class: " номер класса текущего объекта | ∅ |

## Алгоритм метода signal класса cl\_3

#### Функционал: Метод сигнала.

#### Параметры: Адрес строковой переменной. Возвращаемое значение: Отсутствует.

#### Алгоритм метода представлен в таблице 13.

###### Таблица 13 – Алгоритм метода signal класса cl\_3

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal from" координата текущего объекта | 2 |
| 2 |  | Добавление к строке текста "class: " номер класса текущего объекта | ∅ |

## Алгоритм метода signal класса cl\_4

#### Функционал: Метод сигнала.

#### Параметры: Адрес строковой переменной. Возвращаемое значение: Отсутствует.

#### Алгоритм метода представлен в таблице 14.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal from" координата текущего объекта | 2 |
| 2 |  | Добавление к строке текста "class: " номер класса текущего объекта | ∅ |

## Алгоритм метода signal класса cl\_5

#### Функционал: Метод сигнала.

#### Параметры: Адрес строковой переменной. Возвращаемое значение: Отсутствует.

#### Алгоритм метода представлен в таблице 15.

###### Таблица 15 – Алгоритм метода signal класса cl\_5

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal from" координата текущего объекта | 2 |
| 2 |  | Добавление к строке текста "class: " номер класса текущего объекта | ∅ |

## Алгоритм метода signal класса cl\_6

#### Функционал: Метод сигнала.

#### Параметры: Адрес строковой переменной. Возвращаемое значение: Отсутствует.

#### Алгоритм метода представлен в таблице 16.

###### Таблица 16 – Алгоритм метода signal класса cl\_6

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal from" координата текущего объекта | 2 |
| 2 |  | Добавление к строке текста "class: " номер класса текущего объекта | ∅ |

## Алгоритм метода handler класса cl\_1

#### Функционал: Обработчик сигнала. Параметры: строка.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 17.

###### Таблица 17 – Алгоритм метода handler класса cl\_1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal to " координата текущего объекта "Text: "  значение параметра | ∅ |

## Алгоритм метода handler класса cl\_2

#### Функционал: Обработчик сигнала. Параметры: строка.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 18.

###### Таблица 18 – Алгоритм метода handler класса cl\_2

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal to " координата текущего объекта "Text: "  значение параметра | ∅ |

## Алгоритм метода handler класса cl\_3

#### Функционал: Обработчик сигнала. Параметры: строка.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 19.

###### Таблица 19 – Алгоритм метода handler класса cl\_3

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal to " координата текущего объекта "Text: "  значение параметра | ∅ |

## Алгоритм метода handler класса cl\_4

#### Функционал: Обработчик сигнала. Параметры: строка.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 20.

###### Таблица 20 – Алгоритм метода handler класса cl\_4

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal to " координата текущего объекта "Text: "  значение параметра | ∅ |

## Алгоритм метода handler класса cl\_5

#### Функционал: Обработчик сигнала. Параметры: строка.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 21.

###### Таблица 21 – Алгоритм метода handler класса cl\_5

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal to " координата текущего объекта "Text: "  значение параметра | ∅ |

## Алгоритм метода handler класса cl\_6

#### Функционал: Обработчик сигнала. Параметры: строка.

#### Возвращаемое значение: отсутствует. Алгоритм метода представлен в таблице 22.

###### Таблица 22 – Алгоритм метода handler класса cl\_6

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод на экран "Signal to " координата текущего объекта "Text: "  значение параметра | ∅ |

## Алгоритм функции main

#### Функционал: Основная функция. Параметры: отсутствует.

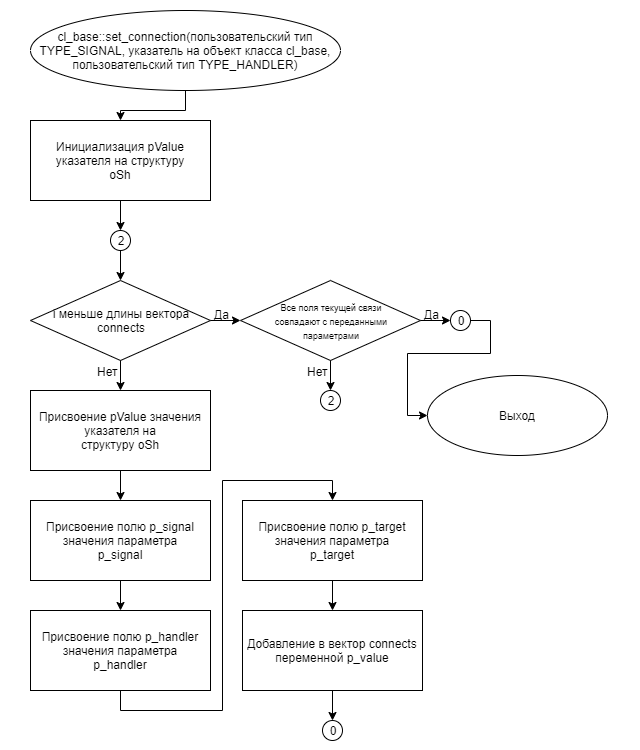
#### Возвращаемое значение: Целый тип - индикация успешной работы. Алгоритм функции представлен в таблице 23.

###### Таблица 23 – Алгоритм функции main

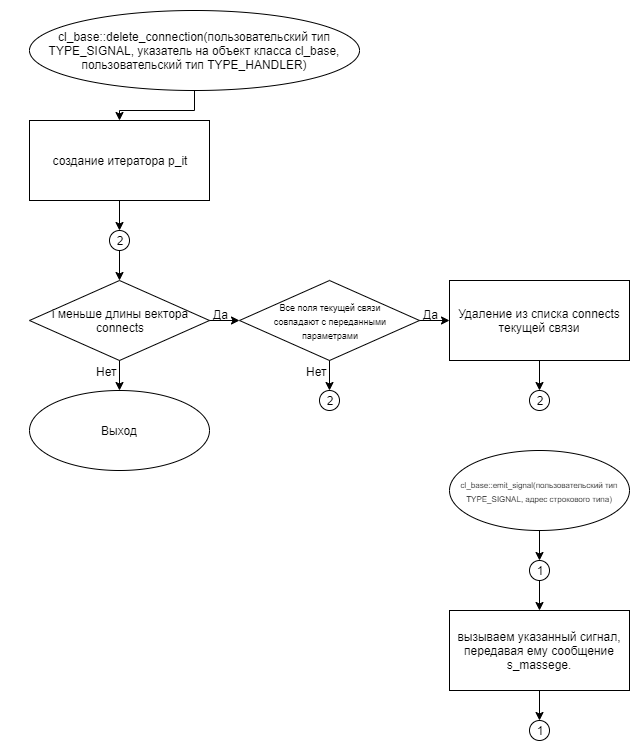
|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Инициализация application | 2 |
| 2 |  | Вызов метода buildTree() | 3 |
| 3 |  | Вызов метода execApp() | ∅ |

# БЛОК-СХЕМЫ АЛГОРИТМОВ

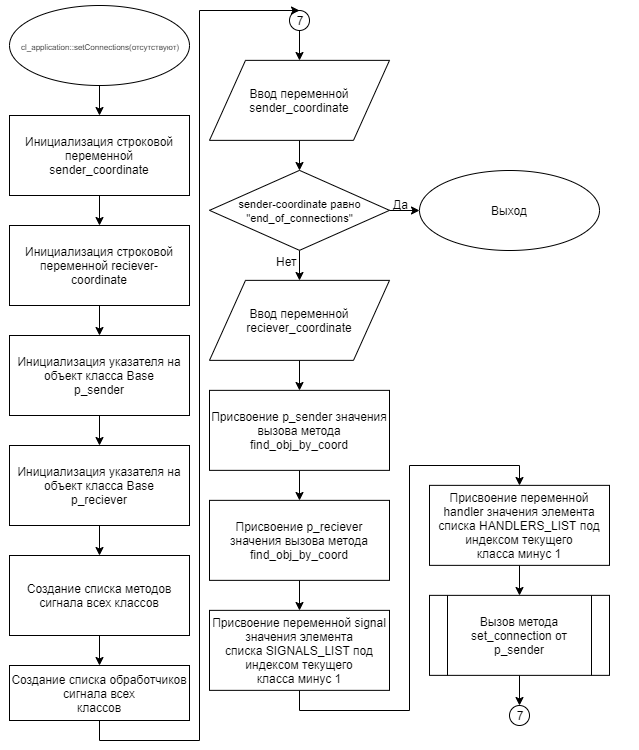
#### Представим описание алгоритмов в графическом виде на рисунках 1-13.



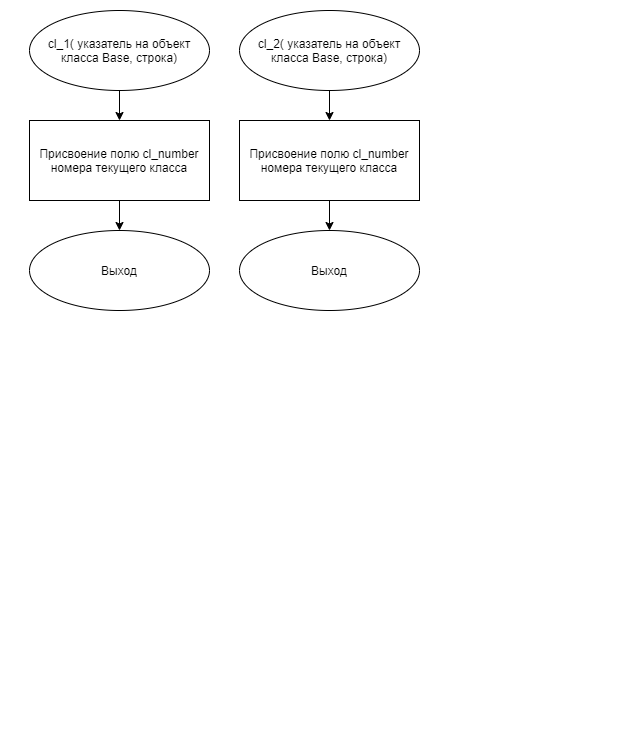
##### Рисунок 1 – Блок-схема алгоритма



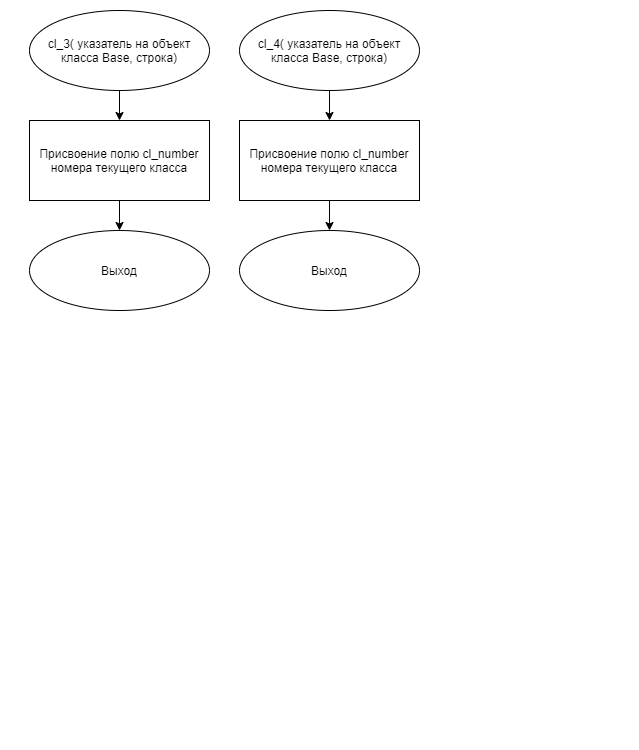
##### Рисунок 2 – Блок-схема алгоритма



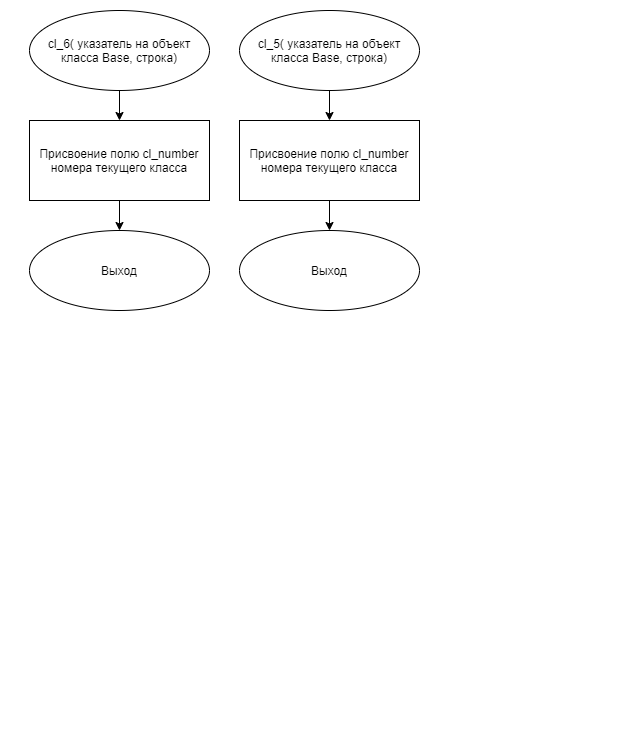
##### Рисунок 3 – Блок-схема алгоритма



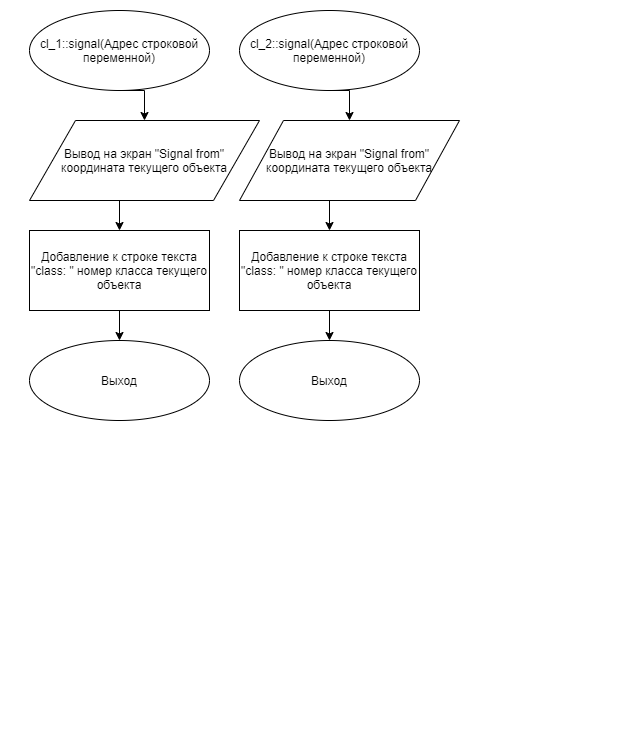
##### Рисунок 4 – Блок-схема алгоритма



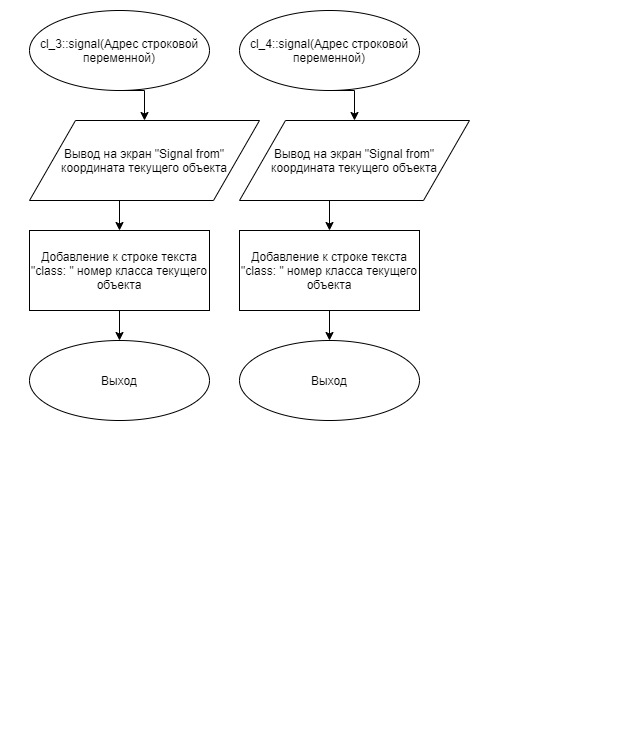
##### Рисунок 5 – Блок-схема алгоритма



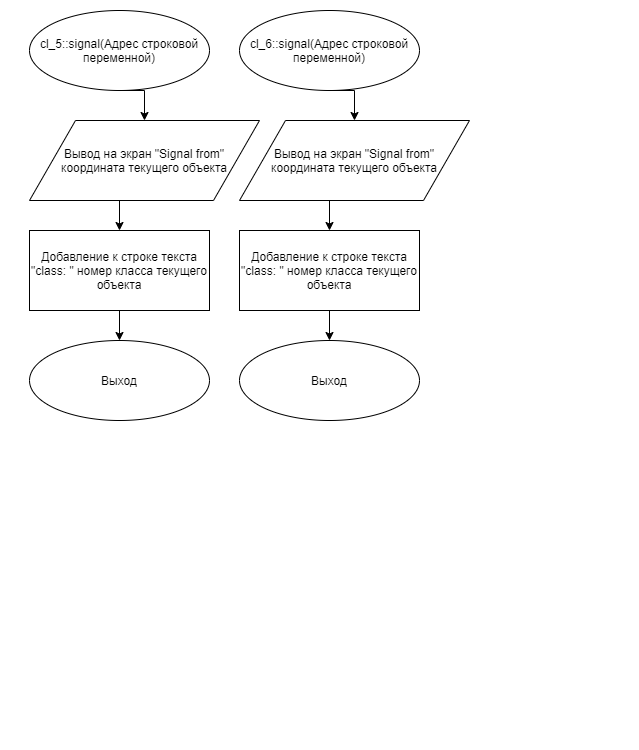
##### Рисунок 6 – Блок-схема алгоритма



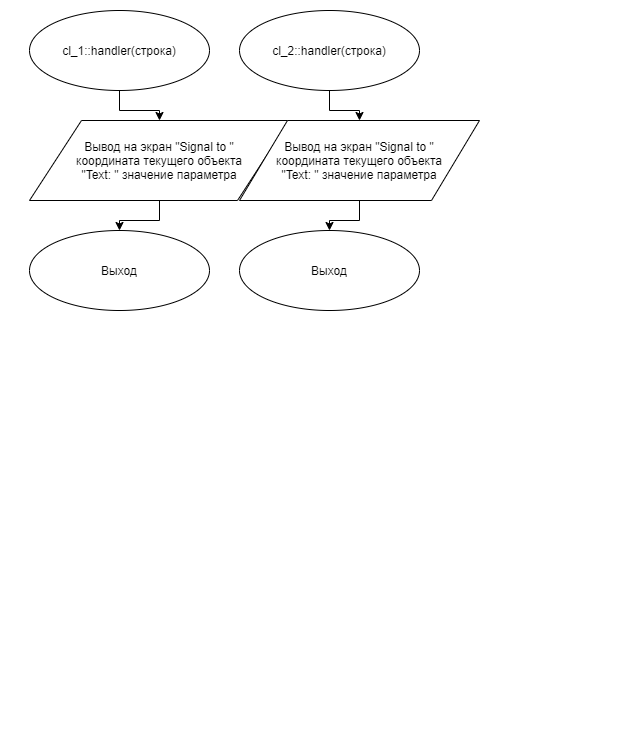
##### Рисунок 7 – Блок-схема алгоритма



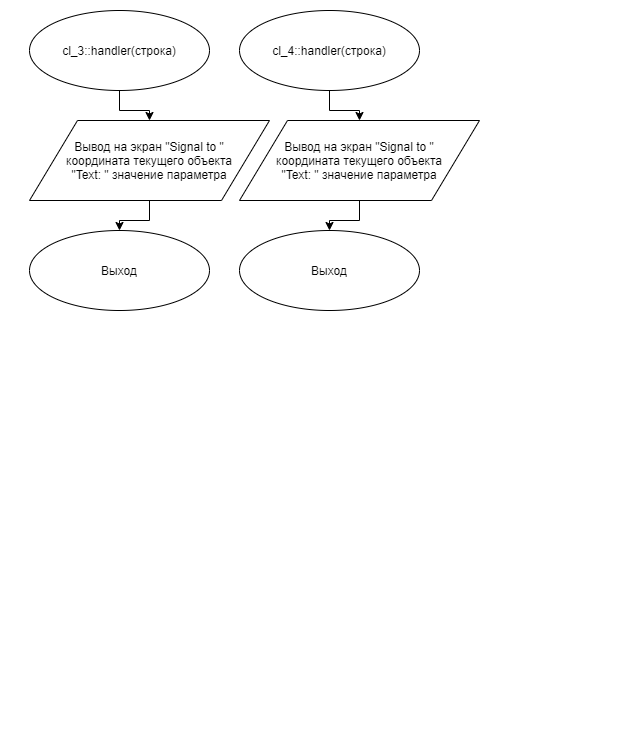
##### Рисунок 8 – Блок-схема алгоритма



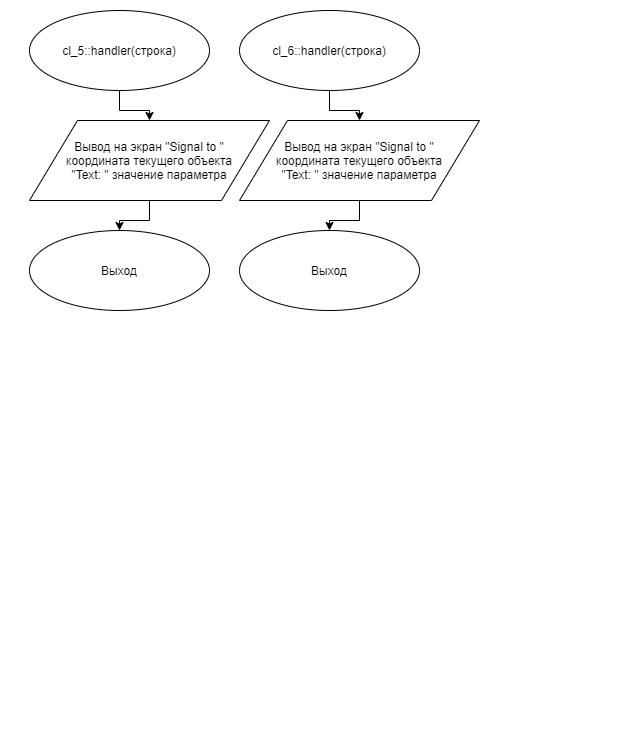
##### Рисунок 9 – Блок-схема алгоритма



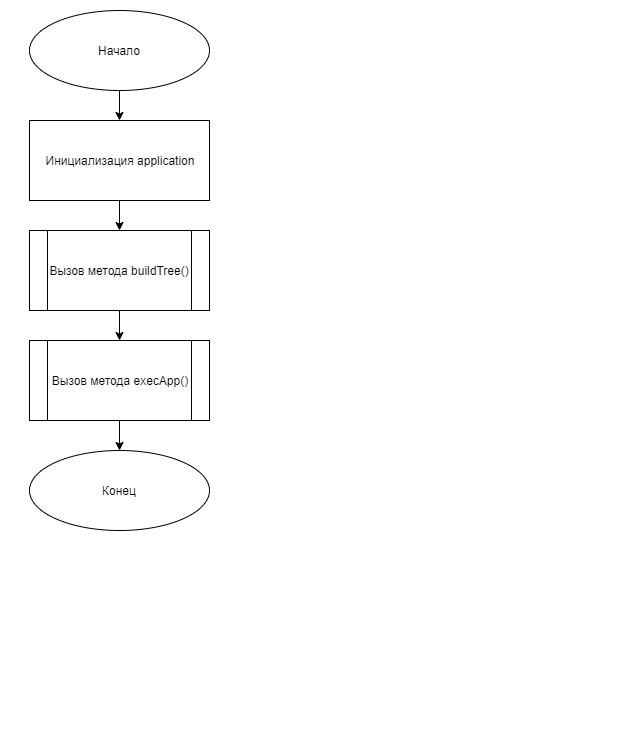
##### Рисунок 10 – Блок-схема алгоритма



##### Рисунок 11 – Блок-схема алгоритма



##### Рисунок 12 – Блок-схема алгоритма



##### Рисунок 13 – Блок-схема алгоритма

# КОД ПРОГРАММЫ

#### Программная реализация алгоритмов для решения задачи представлена

#### ниже.

## Файл cl\_1.cpp

###### Листинг 1 – cl\_1.cpp

#include "cl\_1.h"

cl\_1::cl\_1(cl\_base\* p\_head\_object, string s\_name) :cl\_base(p\_head\_object, s\_name)

{

this->number = 1;

}

void cl\_1::signal\_f(string& msg)

{

cout << endl << "Signal from " << this->get\_path(); msg += " (class: 1)";

}

void cl\_1::handler\_f(string msg)

{

cout << endl << "Signal to " << get\_path() << " Text: " << msg;

}

## Файл cl\_1.h

###### Листинг 2 – cl\_1.h

#ifndef CL\_1\_H #define CL\_1\_H #include "cl\_base.h"

class cl\_1 : public cl\_base

{

public:

cl\_1(cl\_base\* p\_head\_object, string s\_name); void signal\_f(string& msg);

void handler\_f(string msg);

};

#endif

## Файл cl\_2.cpp

###### Листинг 3 – cl\_2.cpp

#include "cl\_2.h"

cl\_2::cl\_2(cl\_base\* p\_head\_object, string s\_name) :cl\_base(p\_head\_object, s\_name)

{

this->number = 2;

}

void cl\_2::signal\_f(string& msg)

{

cout << endl << "Signal from " << this->get\_path(); msg += " (class: 2)";

}

void cl\_2::handler\_f(string msg)

{

cout << endl << "Signal to " << get\_path() << " Text: " << msg;

}

## Файл cl\_2.h

###### Листинг 4 – 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\_name); void signal\_f(string& msg);

void handler\_f(string msg);

};

#endif

## Файл cl\_3.cpp

###### Листинг 5 – cl\_3.cpp

#include "cl\_3.h"

cl\_3::cl\_3(cl\_base\* p\_head\_object, string s\_name) :cl\_base(p\_head\_object, s\_name)

{

this->number = 3;

}

void cl\_3::signal\_f(string& msg)

{

cout << endl << "Signal from " << this->get\_path(); msg += " (class: 3)";

}

void cl\_3::handler\_f(string msg)

{

cout << endl << "Signal to " << get\_path() << " Text: " << msg;

}

## Файл cl\_3.h

###### Листинг 6 – 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\_name); void signal\_f(string& msg);

void handler\_f(string msg);

};

#endif

## Файл cl\_4.cpp

###### Листинг 7 – cl\_4.cpp

#include "cl\_4.h"

cl\_4::cl\_4(cl\_base\* p\_head\_object, string s\_name) :cl\_base(p\_head\_object, s\_name)

{

this->number = 4;

}

void cl\_4::signal\_f(string& msg)

{

cout << endl << "Signal from " << this->get\_path(); msg += " (class: 4)";

}

void cl\_4::handler\_f(string msg)

{

cout << endl << "Signal to " << get\_path() << " Text: " << msg;

}

## Файл cl\_4.h

###### Листинг 8 – 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\_name); void signal\_f(string& msg);

void handler\_f(string msg);

};

#endif

## Файл cl\_5.cpp

###### Листинг 9 – cl\_5.cpp

#include "cl\_5.h"

cl\_5::cl\_5(cl\_base\* p\_head\_object, string s\_name) :cl\_base(p\_head\_object, s\_name)

{

this->number = 5;

}

void cl\_5::signal\_f(string& msg)

{

cout << endl << "Signal from " << this->get\_path(); msg += " (class: 5)";

}

void cl\_5::handler\_f(string msg)

{

cout << endl << "Signal to " << get\_path() << " Text: " << msg;

}

## Файл cl\_5.h

###### Листинг 10 – 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\_name); void signal\_f(string& msg);

void handler\_f(string msg);

};

#endif

## Файл cl\_6.cpp

###### Листинг 11 – cl\_6.cpp

#include "cl\_6.h"

cl\_6::cl\_6(cl\_base\* p\_head\_object, string s\_name) :cl\_base(p\_head\_object, s\_name)

{

this->number = 6;

}

void cl\_6::signal\_f(string& msg)

{

cout << endl << "Signal from " << this->get\_path(); msg += " (class: 6)";

}

void cl\_6::handler\_f(string msg)

{

cout << endl << "Signal to " << get\_path() << " Text: " << msg;

}

## Файл cl\_6.h

###### Листинг 12 – 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\_name); void signal\_f(string& msg);

void handler\_f(string msg);

};

#endif

## Файл cl\_application.cpp

###### Листинг 13 – cl\_application.cpp

#include "cl\_application.h" void setConnections();

cl\_application::cl\_application(cl\_base\* p\_head\_object) : cl\_base(p\_head\_object) {}

void cl\_application::build\_tree\_objects()

{

string s\_head, s\_sub; int s\_num\_obj;

cl\_base\* p\_head = this, \* p\_sub = nullptr; cin >> s\_head;

set\_name(s\_head); while (true)

{

cin >> s\_head;

if (s\_head == "endtree")

{

break;

}

cin >> s\_sub >> s\_num\_obj; if (p\_head != nullptr)

{

p\_head = find\_obj\_by\_coord(s\_head); switch (s\_num\_obj)

{

case 1:

p\_sub = new cl\_1(p\_head, s\_sub); break;

case 2:

p\_sub = new cl\_2(p\_head, s\_sub); break;

case 3:

p\_sub = new cl\_3(p\_head, s\_sub); break;

case 4:

p\_sub = new cl\_4(p\_head, s\_sub); break;

case 5:

p\_sub = new cl\_5(p\_head, s\_sub); break;

case 6:

p\_sub = new cl\_6(p\_head, s\_sub); break;

}

else

{

}

cout << "Object tree";

print\_from\_current();

cout << endl << "The head object " << s\_head << " is not found"; exit(1);

}

}

}

void cl\_application::build\_commands()

{

string line, command, coord, text; vector <TYPE\_SIGNAL> SIGNALS\_LIST =

{

SIGNAL\_D(cl\_1::signal\_f), SIGNAL\_D(cl\_2::signal\_f), SIGNAL\_D(cl\_3::signal\_f), SIGNAL\_D(cl\_4::signal\_f), SIGNAL\_D(cl\_5::signal\_f), SIGNAL\_D(cl\_6::signal\_f)

};

vector<TYPE\_HANDLER> HANDLERS\_LIST =

{

};

while(true)

{

HANDLER\_D(cl\_1::handler\_f), HANDLER\_D(cl\_2::handler\_f), HANDLER\_D(cl\_3::handler\_f), HANDLER\_D(cl\_4::handler\_f), HANDLER\_D(cl\_5::handler\_f), HANDLER\_D(cl\_6::handler\_f)

getline(cin, line);

command = line.substr(0, line.find(' '));

line = line.substr(line.find(' ') + 1, line.size()-1); coord = line.substr(0, line.find(' '));

text = line.substr(line.find(' ')+1); if(command == "END")

{

break;

}

if(line == "")

{

continue;

}

cl\_base\* pSender = this->find\_obj\_by\_coord(coord); if(pSender == nullptr)

{

cout << endl << "Object " << coord << " not found"; continue;

}

if(command == "EMIT")

{

TYPE\_SIGNAL signal = SIGNALS\_LIST[pSender->number - 1]; pSender->emit\_signal(signal, text);

}

if(command == "SET\_CONNECT")

{

cl\_base\* pReceiver = this->find\_obj\_by\_coord(text); if(pReceiver == nullptr)

{

cout<<endl<<"Handler object "<<text<<" not found";

}

TYPE\_SIGNAL signal = SIGNALS\_LIST[pSender->number - 1]; TYPE\_HANDLER handler = HANDLERS\_LIST[pSender->number-1]; pSender->set\_connection(signal, pReceiver, handler);

}

if(command=="DELETE\_CONNECT")

{

cl\_base\* pReceiver = this->find\_obj\_by\_coord(text); if(pReceiver == nullptr)

{

}

else

{

}

}

cout << endl << "Handler object " << text << " not found";

TYPE\_SIGNAL signal = SIGNALS\_LIST[pSender->number-1]; TYPE\_HANDLER handler = HANDLERS\_LIST[pSender->number-1]; pSender->delete\_connection(signal, pReceiver, handler);

if(command=="SET\_CONDITION"){ int state = stoi(text); pSender->setState(state);

}

}

}

int cl\_application::exec\_app()

{

cout << "Object tree"; print\_from\_current(); this->setConnections(); build\_commands(); return 0;

}

void cl\_application::setConnections()

{

string senderCoord; string receiverCoord; cl\_base\* pSender; cl\_base\* pReceiver;

vector<TYPE\_SIGNAL> SIGNALS\_LIST =

{

SIGNAL\_D(cl\_1::signal\_f), SIGNAL\_D(cl\_2::signal\_f), SIGNAL\_D(cl\_3::signal\_f), SIGNAL\_D(cl\_4::signal\_f), SIGNAL\_D(cl\_5::signal\_f), SIGNAL\_D(cl\_6::signal\_f)

};

vector<TYPE\_HANDLER> HANDLERS\_LIST =

{

HANDLER\_D(cl\_1::handler\_f), HANDLER\_D(cl\_2::handler\_f), HANDLER\_D(cl\_3::handler\_f), HANDLER\_D(cl\_4::handler\_f), HANDLER\_D(cl\_5::handler\_f), HANDLER\_D(cl\_6::handler\_f)

};

while(true)

{

cin>>senderCoord;

if(senderCoord == "end\_of\_connections") break; cin >> receiverCoord;

pSender = this->find\_obj\_by\_coord(senderCoord); pReceiver = this->find\_obj\_by\_coord(receiverCoord); TYPE\_SIGNAL signal = SIGNALS\_LIST[pSender->number-1];

TYPE\_HANDLER handler = HANDLERS\_LIST[pReceiver->number-1]; pSender->set\_connection(signal, pReceiver, handler);

}

}

## Файл cl\_application.h

###### Листинг 14 – cl\_application.h

#ifndef CL\_APPLICATION\_H #define CL\_APPLICATION\_H #include "cl\_base.h" #include "cl\_1.h"

#include "cl\_2.h" #include "cl\_3.h" #include "cl\_4.h" #include "cl\_5.h" #include "cl\_6.h"

class cl\_application : public cl\_base

{

public:

cl\_application(cl\_base\* p\_head\_object); void build\_tree\_objects();

int exec\_app();

void build\_commands(); void setConnections(); void handleCommands();

};

#endif

## Файл cl\_base.cpp

###### Листинг 15 – cl\_base.cpp

#include "cl\_base.h"

cl\_base::cl\_base(cl\_base\* p\_head\_object, string s\_name)

{

this->s\_name = s\_name;

this->p\_head\_object = p\_head\_object; if (p\_head\_object != nullptr)

{

p\_head\_object->p\_sub\_objects.push\_back(this);

}

}

cl\_base::~cl\_base()

{

get\_root()->delete\_links(this);

for (int i = 0; i < p\_sub\_objects.size(); i++)

{

delete p\_sub\_objects[i];

}

}

bool cl\_base::set\_name(string s\_new\_name)

{

if (get\_head() != nullptr)

{

for (int i = 0; i < get\_head()->p\_sub\_objects.size(); i++)

{

if (get\_head()->p\_sub\_objects[i]->get\_name() == s\_new\_name)

{

return false;

}

}

}

s\_name = s\_new\_name; return true;

}

void cl\_base::print\_tree(string delay)

{

cout << endl << delay << get\_name(); for (auto p\_sub : p\_sub\_objects)

{

p\_sub->print\_tree(delay + " ");

}

}

void cl\_base::print\_ready(string delay)

{

cout << endl << delay; get\_ready(get\_name());

for (auto p\_sub : p\_sub\_objects)

{

p\_sub->print\_ready(delay + " ");

}

}

string cl\_base::get\_name()

{

return s\_name;

}

cl\_base\* cl\_base::get\_head()

{

return p\_head\_object;

}

cl\_base\* cl\_base::get\_sub\_obj(string s\_name)

{

for (int i = 0; i < p\_sub\_objects.size(); i++)

{

if (p\_sub\_objects[i]->s\_name == s\_name)

{

return p\_sub\_objects[i];

}

}

return nullptr;

}

int cl\_base::count(string name)

{

int count = 0;

if (get\_name() == name)

{

count++;

}

for (int i = 0; i < p\_sub\_objects.size(); i++)

{

count += p\_sub\_objects[i]->count(name);

}

return count;

}

cl\_base\* cl\_base::search\_by\_name(string name)

{

if (s\_name == name)

{

return this;

}

cl\_base\* p\_result = nullptr;

for (int i = 0; i < p\_sub\_objects.size(); i++)

{

p\_result = p\_sub\_objects[i]->search\_by\_name(name); if (p\_result != nullptr)

{

return p\_result;

}

}

return nullptr;

}

cl\_base\* cl\_base::search\_cur(string name)

{

if (count(name) != 1)

{

return nullptr;

}

return search\_by\_name(name);

}

cl\_base\* cl\_base::search\_from\_root(string name)

{

if (p\_head\_object != nullptr)

{

}

else

{

}

}

return p\_head\_object->search\_from\_root(name);

return search\_cur(name);

void cl\_base::set\_ready(int s\_new\_ready)

{

if (s\_new\_ready != 0)

{

if (p\_head\_object == nullptr || p\_head\_object != nullptr && p\_head\_object->p\_ready != 0)

{

p\_ready = s\_new\_ready;

}

}

else

{

p\_ready = s\_new\_ready;

for (int i = 0; i < p\_sub\_objects.size(); i++)

{

p\_sub\_objects[i]->set\_ready(s\_new\_ready);

}

}

}

void cl\_base::get\_ready(string name)

{

if (get\_name() == name)

{

if (p\_ready != 0)

{

cout << get\_name() << " is ready";

}

else

{

}

}

else

{

cout << get\_name() << " is not ready";

for (int i = 0; i < p\_sub\_objects.size(); i++)

{

return p\_sub\_objects[i]->get\_ready(name);

}

}

}

bool cl\_base::change\_head\_obj(cl\_base\* new\_head\_obj)

{

if (new\_head\_obj != nullptr)

{

cl\_base\* temp = new\_head\_obj; while (temp != nullptr)

{

temp = temp->p\_head\_object; if (temp == this)

{

return false;

}

}

if (new\_head\_obj->get\_sub\_obj(get\_name()) == nullptr && p\_head\_object != nullptr)

{

p\_head\_object->p\_sub\_objects.erase(find(p\_head\_object-

>p\_sub\_objects.begin(), p\_head\_object->p\_sub\_objects.end(), this)); new\_head\_obj->p\_sub\_objects.push\_back(this); p\_head\_object = new\_head\_obj;

return true;

}

}

return false;

}

void cl\_base::delete\_subordinate\_obj(string name)

{

cl\_base\* subordinate\_obj = get\_sub\_obj(name); if (subordinate\_obj != nullptr)

{

p\_sub\_objects.erase(find(p\_sub\_objects.begin(), p\_sub\_objects.end(), subordinate\_obj));

delete subordinate\_obj;

}

}

cl\_base\* cl\_base::find\_obj\_by\_coord(string s\_object\_path)

{

if (s\_object\_path == "")

{

return nullptr;

}

cl\_base\* head\_obj = this; string s\_path\_item;

if (s\_object\_path == "." || s\_object\_path == "/")

{

return head\_obj;

}

if (s\_object\_path[0] == '.')

{

s\_object\_path.erase(s\_object\_path.begin()); return search\_by\_name(s\_object\_path);

}

if (s\_object\_path[1] == '/' && s\_object\_path[0] == '/')

{

s\_object\_path.erase(s\_object\_path.begin()); s\_object\_path.erase(s\_object\_path.begin()); return this->search\_from\_root(s\_object\_path);

}

if (s\_object\_path[0] == '/')

{

s\_object\_path.erase(s\_object\_path.begin()); while (head\_obj->p\_head\_object != nullptr)

{

head\_obj = head\_obj->p\_head\_object;

}

}

stringstream ss\_path(s\_object\_path); while (getline(ss\_path, s\_path\_item, '/'))

{

head\_obj = head\_obj->get\_sub\_obj(s\_path\_item); if (head\_obj == nullptr)

{

return nullptr;

}

}

return head\_obj;

}

void cl\_base::print\_from\_current(int n)

{

cout << endl;

for (int i = 0; i < n; i++)

{

cout << " ";

}

cout << s\_name;

for (auto p\_subordinate\_object : p\_sub\_objects)

{

p\_subordinate\_object->print\_from\_current(n + 1);

}

}

void cl\_base::set\_connection(TYPE\_SIGNAL p\_signal, cl\_base\* p\_target, TYPE\_HANDLER p\_handler)

{

o\_sh\* p\_value;

for (int i = 0; i < connects.size(); i++)

{

if (connects[i]->p\_signal == p\_signal && connects[i]->p\_handler == p\_handler && connects[i]->p\_target == p\_target)

{

return;

}

}

p\_value = new o\_sh();

p\_value->p\_signal = p\_signal; p\_value->p\_handler = p\_handler; p\_value->p\_target = p\_target;

connects.push\_back(p\_value);

}

void cl\_base::delete\_connection(TYPE\_SIGNAL p\_signal, cl\_base\* p\_target, TYPE\_HANDLER p\_handler)

{

vector<o\_sh\*>::iterator p\_it;

for (p\_it = connects.begin(); p\_it != connects.end(); p\_it++)

{

if ((\*p\_it)->p\_signal == p\_signal && (\*p\_it)->p\_target == p\_target /\*&& (\*p\_it)->p\_handler == p\_handler\*/)

{

delete\* p\_it;

p\_it = connects.erase(p\_it); p\_it--;

}

}

}

void cl\_base::emit\_signal(TYPE\_SIGNAL p\_signal, string s\_massege)

{

if (p\_ready != 0)

{

TYPE\_HANDLER pHandler; cl\_base\* pObject;

(this->\*p\_signal)(s\_massege);

for (int i = 0; i < connects.size(); i++)

{

if (connects[i]->p\_signal == p\_signal)

{

pHandler = connects[i]->p\_handler; pObject = connects[i]->p\_target; if (pObject->p\_ready != 0)

{

(pObject->\*pHandler)(s\_massege);

}

}

}

}

}

string cl\_base::get\_path()

{

cl\_base\* p\_head\_object = this->get\_head(); if (p\_head\_object != nullptr)

{

if (p\_head\_object->get\_head() == nullptr)

{

}

else

{

}

}

return p\_head\_object->get\_path() + s\_name;

return p\_head\_object->get\_path() + "/" + s\_name;

return "/";

}

void cl\_base::setState(int state)

{

if (state == 0)

{

this->p\_ready = 0;

for (int i = 0; i < p\_sub\_objects.size(); i++) { p\_sub\_objects[i]->setState(0);

}

return;

}

if (this->p\_head\_object == nullptr || this->p\_head\_object->p\_ready != 0) { this->p\_ready = state;

}

}

void cl\_base::delete\_links(cl\_base\* targ)

{

for (auto p\_it = connects.begin(); p\_it != connects.end(); p\_it++)

{

if ((\*p\_it)->p\_target == targ)

{

delete (\*p\_it); connects.erase(p\_it); p\_it--;

}

}

for (auto p\_sub : p\_sub\_objects)

{

p\_sub->delete\_links(targ);

}

}

cl\_base\* cl\_base::get\_root()

{

if (p\_head\_object != nullptr)

{

p\_head\_object->get\_root();

}

return this;

}

## Файл cl\_base.h

###### Листинг 16 – cl\_base.h

#ifndef CL\_BASE\_H #define CL\_BASE\_H #include <iostream> #include <string> #include <vector> #include <sstream> #include <algorithm>

#define SIGNAL\_D(signal\_f) (TYPE\_SIGNAL)(&signal\_f) #define HANDLER\_D(handler\_f) (TYPE\_HANDLER)(&handler\_f) using namespace std;

class cl\_base;

typedef void (cl\_base::\* TYPE\_SIGNAL) (string& msg); typedef void (cl\_base::\* TYPE\_HANDLER) (string msg);

struct o\_sh

{

TYPE\_SIGNAL p\_signal; TYPE\_HANDLER p\_handler; cl\_base\* p\_target;

};

class cl\_base

{

private:

string s\_name;

cl\_base\* p\_head\_object;

vector <cl\_base\*> p\_sub\_objects; int p\_ready = 1;

vector<o\_sh\*> connects; public:

cl\_base(cl\_base\* p\_head\_object, string s\_name = "Base Object"); bool set\_name(string s\_new\_name);

string get\_name(); cl\_base\* get\_head();

void print\_tree(string delay = ""); cl\_base\* get\_sub\_obj(string s\_name);

~cl\_base();

int count(string name);

cl\_base\* search\_by\_name(string name); cl\_base\* search\_cur(string name); cl\_base\* search\_from\_root(string name); void set\_ready(int s\_new\_ready);

void get\_ready(string name);

void print\_ready(string delay = "");

bool change\_head\_obj(cl\_base\* new\_head\_obj); void delete\_subordinate\_obj(string name);

cl\_base\* find\_obj\_by\_coord(string s\_object\_path); void print\_from\_current(int n = 0);

void set\_connection(TYPE\_SIGNAL p\_signal, cl\_base\* p\_target, TYPE\_HANDLER p\_handler);

void delete\_connection(TYPE\_SIGNAL p\_signal, cl\_base\* p\_target, TYPE\_HANDLER p\_handler);

void emit\_signal(TYPE\_SIGNAL p\_signal, string massege); string get\_path();

int number = 1;

typedef void (cl\_base::\* TYPE\_SIGNAL)(string&); typedef void (cl\_base::\* TYPE\_HANDLER)(string); void setState(int state);

void delete\_links(cl\_base\* targ); cl\_base\* get\_root();

};

#endif

## Файл main.cpp

###### Листинг 17 – main.cpp

#include "cl\_application.h" int main()

{

cl\_application ob\_cl\_application ( nullptr ); ob\_cl\_application.build\_tree\_objects ( ); return ob\_cl\_application.exec\_app ( );

}

# ТЕСТИРОВАНИЕ

#### Результат тестирования программы представлен в таблице [24](#_bookmark48).

###### Таблица 24 – 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:  /object\_s2/object\_s4 (class: 4)  Signal to / Text:  /object\_s2/object\_s4 (class: 4)  Object message not found  Signal from  /object\_s2/object\_s4 Signal to  /object\_s2/object\_s6 Text:  /object\_s2/object\_s4 (class: 4)  Signal to / Text:  /object\_s2/object\_s4 (class: 4)  Object message not found  Signal from  /object\_s2/object\_s4 Signal to  /object\_s2/object\_s6 Text:  /object\_s2/object\_s4 (class: 4)  Signal to / Text:  /object\_s2/object\_s4 (class: 4)  Object message not | 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:  /object\_s2/object\_s4 (class: 4)  Signal to / Text:  /object\_s2/object\_s4 (class: 4)  Object message not found  Signal from  /object\_s2/object\_s4 Signal to  /object\_s2/object\_s6 Text:  /object\_s2/object\_s4 (class: 4)  Signal to / Text:  /object\_s2/object\_s4 (class: 4)  Object message not found  Signal from  /object\_s2/object\_s4 Signal to  /object\_s2/object\_s6 Text:  /object\_s2/object\_s4 (class: 4)  Signal to / Text:  /object\_s2/object\_s4 (class: 4)  Object message not |

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемые выходные данные** | **Фактические выходные данные** |
|  | found | found |
| Signal from | Signal from |
| /object\_s1 | /object\_s1 |
| Object 4 not found | Object 4 not found |

# ЗАКЛЮЧЕНИЕ

Изучение курса "Объектно-ориентированное программирование" было для меня приятным открытием в программировании. Я получил понимание методологии объектно-ориентированного программирования и научился использовать классы и объекты для создания системных архитектур на высоком уровне. В ходе выполнения различных задач по данной дисциплине, я убедился, насколько объектно-ориентированное программирование упрощает написание больших программ. В ходе обучения, я освоил такие базовые концепции, как инкапсуляция, наследование и полиморфизм. При изучении курса использовался язык программирования C++, что помогло мне овладеть различными концепциями, такими как указатели, ссылки, абстрактные классы, перегрузка и переопределение функций, перегрузка унарных и бинарных операций, структуры и контейнеры.

Пройденный курс помог мне лучше понять, как работает программное обеспечение, и как важно использовать объектно-ориентированный подход для разработки масштабных проектов. Теперь я готов более уверенно использовать объектно-ориентированный подход в моей будущей работе.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

#### ГОСТ 19 Единая система программной документации.

#### 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\_ra bot\_3.pdf (дата обращения 05.05.2021).

#### 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).

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

#### — 624 с.

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

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