Постановка задачи

Реализация сигналов и обработчиков

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

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).
Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:
1. Установления связи между сигналом текущего объекта и обработчиком целевого объекта;
2. Удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
3. Выдачи сигнала от текущего объекта с передачей строковой переменной.

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

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

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в контрольной работе № 1.
Система содержит объекты трех классов с номерами: 1,2,3. Классу корневого объекта соответствует номер 1.
В каждом классе реализован один метод сигнала и один метод обработчика.
Реализовать алгоритм работы системы:
1. В методе построения дерева иерархи объектов:

1.1. Построение иерархии объектов согласно вводу.
1.2. Ввод и построение множества связей сигнал-обработчик для заданных пар объектов.
2. В методе отработки программы:
2.1. Цикл до признака завершения ввода.
2.1.1. Ввод наименования объекта и текста сообщения.
2.1.2. Вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной содержащей текст сообщения.
2.2. Конец цикла.

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

Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в фрагменте методического указания в контрольной работе № 1.

После ввода состава дерева иерархии построчно вводится:
«уникальный номер связи»«наименование объекта выдающей сигнал»«наименование целевого объекта»
Уникальный номер связи – натуральное число.
Ввод информации для построения связей завершается строкой, которая содержит 0.

После завершения ввода связей построчно вводиться:
«наименование объекта выдающей сигнал»«текст сообщения из одного слова без пробелов»
Последняя строка ввода содержит слово:
endsignals

Описание выходных данных

Первая строка:
Object tree

Со второй строки вывести иерархию построенного дерева. Следующая после вывода дерева объектов строка содержит: Set connects

Далее, построчно:
«уникальный номер связи»«наименование объекта выдающей сигнал»«наименование целевого объекта»
Последовательность вывода совпадает с последовательностью ввода связей. Разделитель один пробель.

Следующая после вывода информации о связях объектов строка содержит: Emit signals

Далее, построчно:
Signal to «наименование целевого объекта» Text: «наименование объекта выдающей сигнал» -> «текст сообщения из одного слова без пробелов» Разделитель один пробель.

Метод решения

Заголовок класса

Заголовок метода

Оператор цикла

Условный оператор

Иначе

Присвоение

Инкрементный оператор

Библиотека set

Библеотека vector

(Красным отмечено, что изменилось/добавилось по сравнению с прошлой лабораторной работы)

Класс Service

Свойства класса Service:

- private:
  - static unsigned longUUID_seq - последовательность для уник. индентификаторов
  - unsigned long UUID - уник. индентификатор сервиса
  - string name - имя сервиса
  - int classId - номер класса
  - int status - состояние сервиса
  - vector<Service*> children - дочерние объекты

public:

6.const string TAB = " " - 4 пробела

Методы класса Service:

- private:
  - static string getPathItem(string adress, int level) - получение имени объекта из адреса, находящееся на позиции level
- public:
  - Service(const string name, const int classId, const int status) - параметризованный конструктор класса Service
  - int addChild(const string parentName, Service* service) - добавление объекта виерархию
  - Service* findByUniqueName(const string name) - поиск объекта в иерархии по имени
  - void getTree() - вывод иерархии
  - void getFineTree(int level = 0) - вывод иерархии с учётом форматирования Service* findByAdress(const string) - поиск сервиса в дереве по адресу
  - unsigned long getUUID() - Getter UUID
  - string getName() - Getter name
  - int getClassId() - Getter classId
  - int getStatus() - Getter status

Класс App

Свойства класса App:

- private:
  - const string END_WORD - строка, которая заканчивает ввод иерархии
  - const string END_WORD2 - строка, которая заканчивает ввод проверяемых адресов
  - const char END_WORD_CONNECTIONS - число, которое заканчивает ввод

соединений сигналов и обработчиков
- const string END_WORD_SIGNALS - строка, которая заканчивает ввод подаваемых сигналов
- vector<string> adresses - вектор проверяемых векторов
- vector<pair<string, string>> signals - список сигналов
- vector<pair<int, pair<string,string>>> connectionsToPrint - список соединений для удобного вывода

Методы класса App:

- public:
  - App(const string name) - Параметризованный конструктор класса App
  - void initialize() - Метод для создания иерархии и ввода искомых адресов
  - void initializeSearch() - Метод для ввода искомых проверяемых адресов
  - void initializeConnections() - Метод для ввода соединений сигналов и обработчиков
  - void initalizeSignals() - Метод для ввода сигналов
  - void getSearchResult() - Метод для получения результата поиска по искомым адресам
  - void getConnectionList() - Метод для вывода списка соединений
  - void getSignalsResult() - Метод для выполнения и вывода результата работы обработчиков
  - void input() - Метод инициализации, всех вводы
  - void output() - Метод результата работы, все выводы
  - void handler(string msg) - Метод-обработчик
  - void sendSignal(string& msg) - Метод-отправлятель-сигналов

Класс Servlet

Свойства класса Servlet:

private:

- string serverIp - адрес сервера

Методы класса Servlet:

public:

- Servlet(const string name, const int status, const string serverIp) - Параметризованный конструктор класса Servlet
- string getServerIp() - Getter serverIp
- void handler(string msg) - Метод-обработчик
- void sendSignal(string& msg) - Метод-отправлятель-сигналов

Класс Asset

Свойства класса Asset:

private:

- int size - размер в байтах

Методы класса Asset:

- Asset(const string name, const int status, const int size = 0) - параметризованныйконструктор класса Asset
- public int getSize() - Getter size
- void handler(string msg) - Метод-обработчик
- void sendSignal(string& msg) - Метод-отправлятель-сигналов

typedef void (SignalBase ::* TYPE_SIGNAL) (string&);

typedef void (SignalBase ::* TYPE_HANDLER) (string);

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)

#define HANDLER_D(handler_f) (TYPE_HANDLER) (&handler_f)

Структура Connection:

- unsigned long id
- TYPE_SIGNAL p_signal
- SignalBase* p_signalBase
- TYPE_HANDLE p_handler
- string emitterAdress
- string handlerAdress
- friend bool operator==(Connection&, Connection)

Класс SignalBase

Свойства класса SignalBase:

private:

- vector<Connection*> connections - соединения

Методы клаcа SignalBase:

public:

- int emitSignal(TYPE_SIGNAL, string&) - метод отправления сигнала
- int removeConnection(TYPE_SIGNAL, SignalBase*, TYPE_HANDLER) - метод удаления соединения

- Connection* findLikeInConnections(Connection& c) - метод поиска одинакового соединения в списке connections
- Connection* findFirstWithTYPE_SIGNAL(TYPE_SIGNAL) - метод поиска соединения по сигналу
- vector<Connection*> getConnections();

Описание алгоритма

Функция: main

Функционал: Основная функция

Параметры: нет

Возвращаемое значение: int, код ошибки

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта root класса App конструктором по умолчанию	2	
2		Вызов метода input объекта root	3	
3		Вызов метода output объекта root	∅	

Класс объекта: App

Модификатор доступа: public

Метод: initializeConnections

Функционал: Метод для ввода соединений сигналов и обработчиков

Параметры: нет

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление unsigned long id string emitterName, handlerName Инициализация: Service* emitter = 0, *handler = 0	2	
2		Начало цикла	3	Цикл для ввода соединений обработчиков и

				сигналов
3		Ввод id	4	
4	id не равно END_WORD_CONNECTIONS		5	END_WORD_CONNECTIONS = 0
	В обратном случае	Выход из цикла	∅	
5		Ввод emitterName, handlerName	6	
6		Вызов метода findByAdress объекта App с аргументом (emitterName) и (handlerName), положить результат в emitter и handler соответственно	7	
7		Инициализация TYPE_SIGNAL signal_d = 0	8	
8		Начало switch(результат выполнения getClassId объекта emitter)	9	
9	=1	Присвоение signal_d = указателю на sendSignal класса App	10	
	=2	Присвоение signal_d = указателю на sendSignal класса Servlet	10	
	=3	Присвоение signal_d = указателю на sendSignal класса Asset	10	
10		Инициализация TYPE_HANDLER handler_d =	11	
11		Начало switch(результат выполнения getClassId объекта handler)	12	
12	=1	Присвоение handler_d = указателю на handler класса App	13	
	=2	Присвоение handler_d = указателю на handler класса Servlet	13	
	=3	Присвоение handler_d = указателю на handler класса Asset	13	
13		Вызов метода addConnection	14	

		объекта emitter с аргументами (id, emitterName, handlerName, signal_d, handler, handler_d)		
14		Добавление соединения в connectionsToPrint	3	

Класс объекта: App

Модификатор доступа: public

Метод: initializeSignals

Функционал: Метод для ввода сигналов

Параметры: нет

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		Ввод сигналов согласно формату, тем самым заполненяя signals до ввода endsignals	∅	

Класс объекта: App

Модификатор доступа: public

Метод: getConnectionList

Функционал: Метод для вывода списка соединений

Параметры: нет

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		Перебор всех элементов вектора connectionToPrint и вывод каждого соединения согласно форматированию	∅	

Класс объекта: App

Модификатор доступа: public

Метод: getSignalsResult

Функционал: Метод для выполнения и вывода результата работы обработчико

Параметры: нет

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1	signals не пуст	Вывод "\nEmit signals" Перебор всех элементов pair<string,string>kv вектора signals	2	
	В другом случае		∅	
2	Перебор kv не закончен	следующий kv	3	
	В другом случае	Выход из перебора	∅	
3		Получение string emitterName и string command из kv	4	
4		Вызов метода findByAdress(emitterName), положим результат в emitter	5	Получение эмиттера по адресу
5		Выбор emit_signal какого класса для инициализации TYPE_SIGNAL signal_d в зависимости от свойства classId объекта emitter	6	
6		Добавление в команду имени эмиттера как аргумент	7	
7		Вызов метода emitSignal(signal_d, command) объекта emitter	2	

Класс объекта: App_Servlet_Asset

Модификатор доступа: public

Метод: handler

Функционал: Метод-обработчик

Параметры: string, msg, сообщение

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		Получение первого слова из msg и записывание его в emitterName	2	через поток stringstream ss
2		Стирание первого слова из msg	3	
3		Получение string handlerName из результата вызова метода getName() этого объекта	4	

4		Вывод сигнала согласно форматирования	∅	

Класс объекта: App_Servlet_Asset

Модификатор доступа: public

Метод: sendSignal

Функционал: Метод-отправлятель-сигналов

Параметры: string&, msg, отправляемое сообщение

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода emitSignal(преобразованный указатель на метод эмиттера через SIGNAL_D, msg) этого объекта	∅	

Функция: operator==

Функционал: Опреация сравнения

Параметры: Connection& c1, Connection& c2 - два сравниваемых соединения

Возвращаемое значение: bool, одинаковы ли они?

№	Предикат	Действия	№ перехода	Комментарий
1	Не совпадают хоть по одному свойству c1 и c2	Вернуь ложь	∅	
	В другом случае		2	
2		Вернуть ложь	∅	

Класс объекта: SignalBase

Модификатор доступа: public

Метод: emitSignal(TYPE_SIGNAL, string&)

Функционал: метод отправления сигнала

Параметры: TYPE_SIGNAL, signal, сигнал; string&, command, сообщение в сигнале

Возвращаемое значение: int, код ошибки

№	Предикат	Действия	№ перехода	Комментарий
1		Перебор всех элементов Connection* connection в векторе connections	2	
2	перебор не	Следующий connection	3	

	закончен			
	В другом случае		∅	
3		Получение TYPE_SIGNAL p_signal из connection	4	
4	p_signal совпадает с signal		5	
	В другом случае		2	
5		Получение SignalBase* p_signalBase и TYPE_HANDLER p_handler из connection	6	
6		Вызов метода по указателю p_handler(command) из объекта p_signalBase	2	

Класс объекта: SignalBase

Модификатор доступа: public

Метод: removeConnection

Функционал: метод удаления соединения

Параметры: TYPE_SIGNAL, signal, сигнал; SignalBase*, obj, объект; TYPE_HANDLER, handler, метод-обработчик

Возвращаемое значение: int, код ошибки

№	Предикат	Действия	№ перехода	Комментарий
1		Поиск идентичного по параметрам из функции соединения и его удаление	∅	

Класс объекта: SignalBase

Модификатор доступа: public

Метод: findLikeInConnections

Функционал: метод поиска одинакового соединения в списке connections

Параметры: Connection&, c, соединение

Возвращаемое значение: Connection*, указатель на идентичное к (c) соединение в списке

№	Предикат	Действия	№ перехода	Комментарий

1		Поиск и возвращение идентичного соединения в списке connections(иначе 0)	∅	

Класс объекта: SignalBase

Модификатор доступа: public

Метод: findFirstWithTYPE_SIGNAL

Функционал: метод поиска соединения по сигналу

Параметры: TYPE_SIGNAL signal, сигнал

Возвращаемое значение: Connection*

№	Предикат	Действия	№ перехода	Комментарий
1		Поиск и возврат первого соединения, у которого совпадает свойство p_signal с signal(иначе 0)	∅	

Класс объекта: SignalBase

Модификатор доступа: public

Метод: getConnections

Функционал: GETTER connections

Параметры: нет

Возвращаемое значение: vector<Connection*>, соединения

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат копии connections	∅	

Класс объекта: App

Модификатор доступа: public

Метод: input

Функционал: Инициализация, все вводы

Параметры: нет

Возвращаемое значение: void

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода initalize этого объекта	2	
2		Вызов метода initializeConnections этого	3	

		объекта		
3		Вызов метода initializeSignals этого объекта	∅	

Класс объекта: App

Модификатор доступа: public

Метод: output

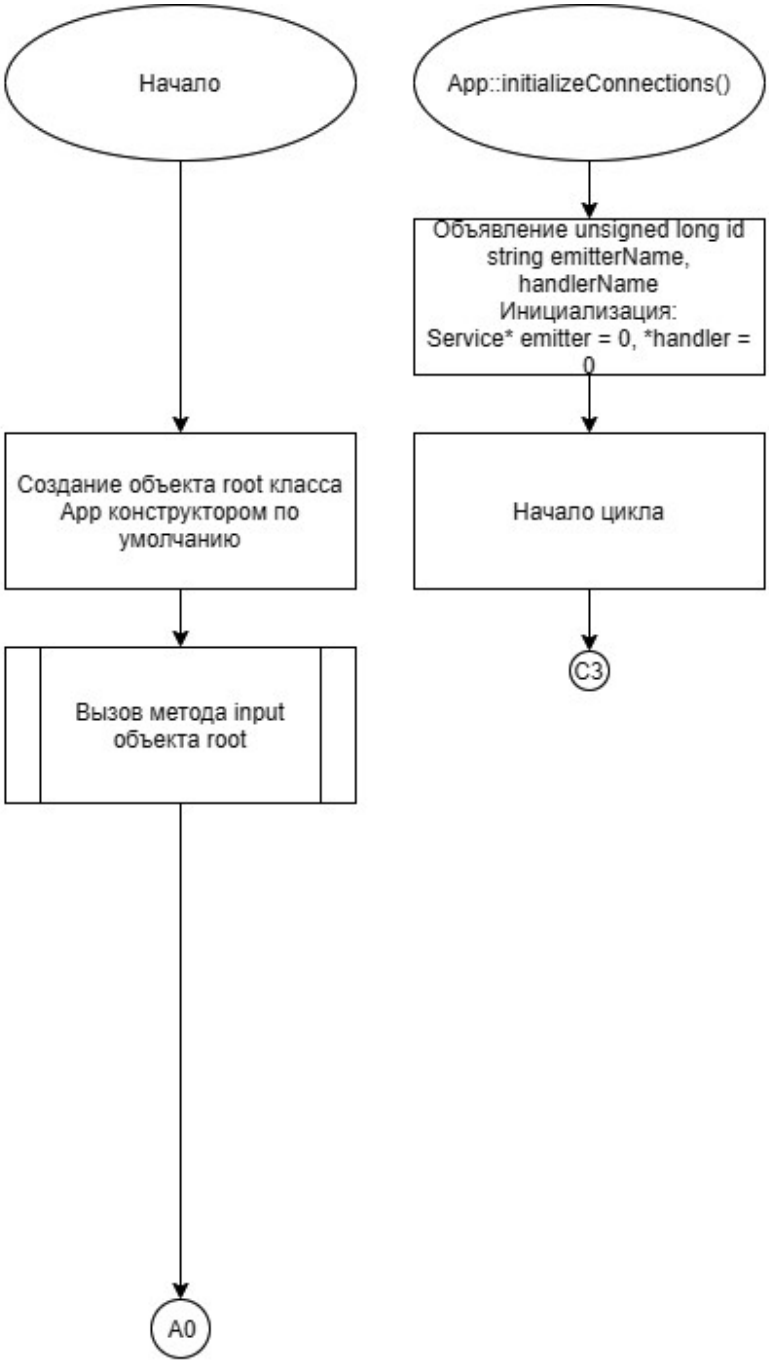
Функционал: Метод результата работы, все выводы

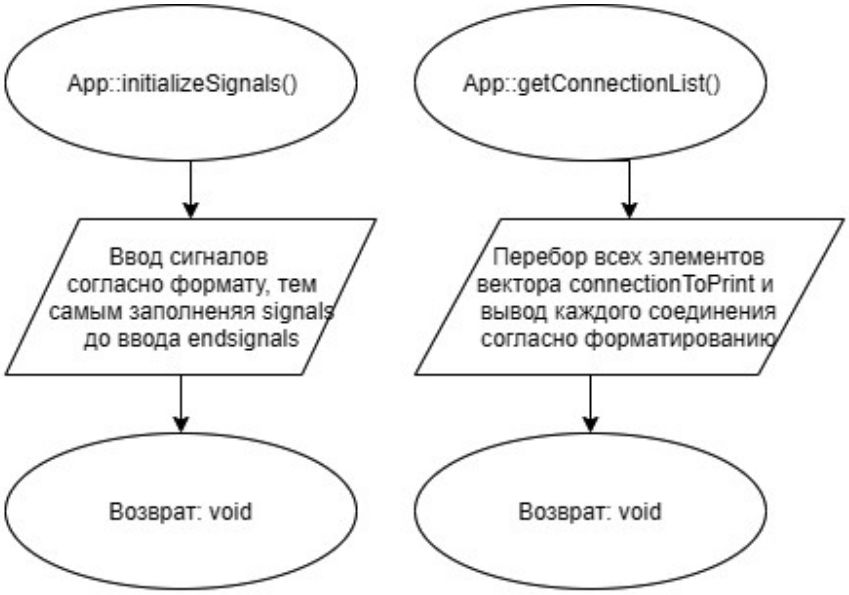
Параметры: нет

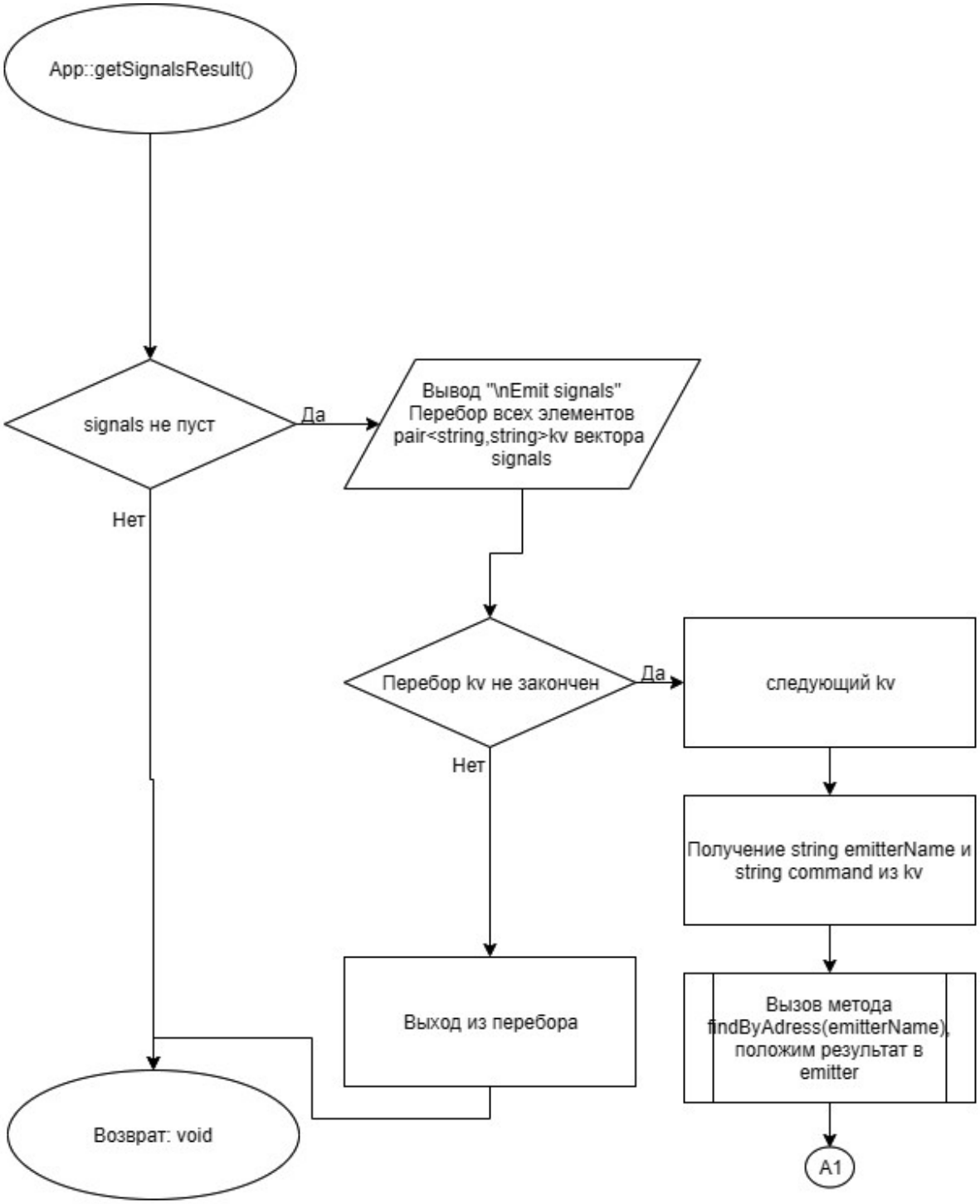
Возвращаемое значение: void

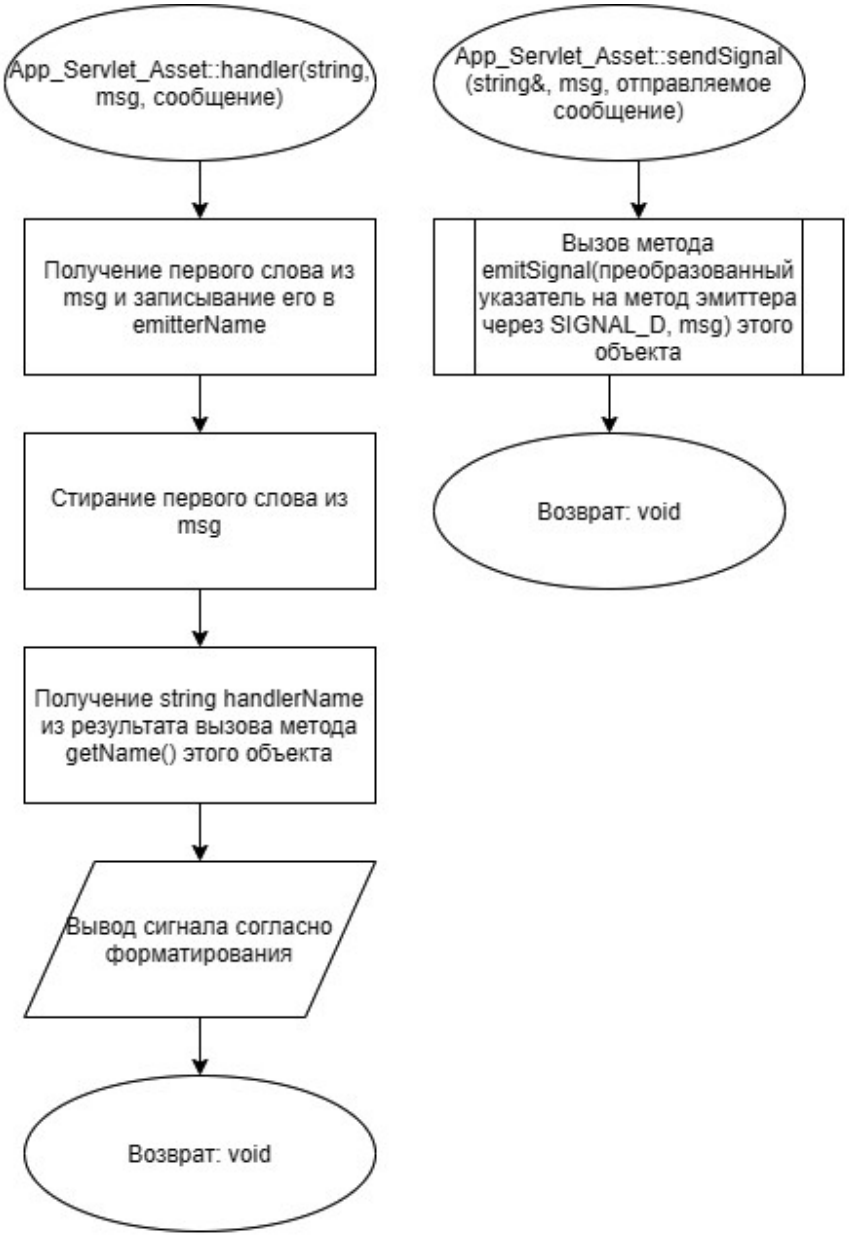
№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода getFineTree этого объекта	2	
2		Вызов метода getConnectionList этого объекта	3	
3		Вызов метода getSignalsResult этого объекта	∅	

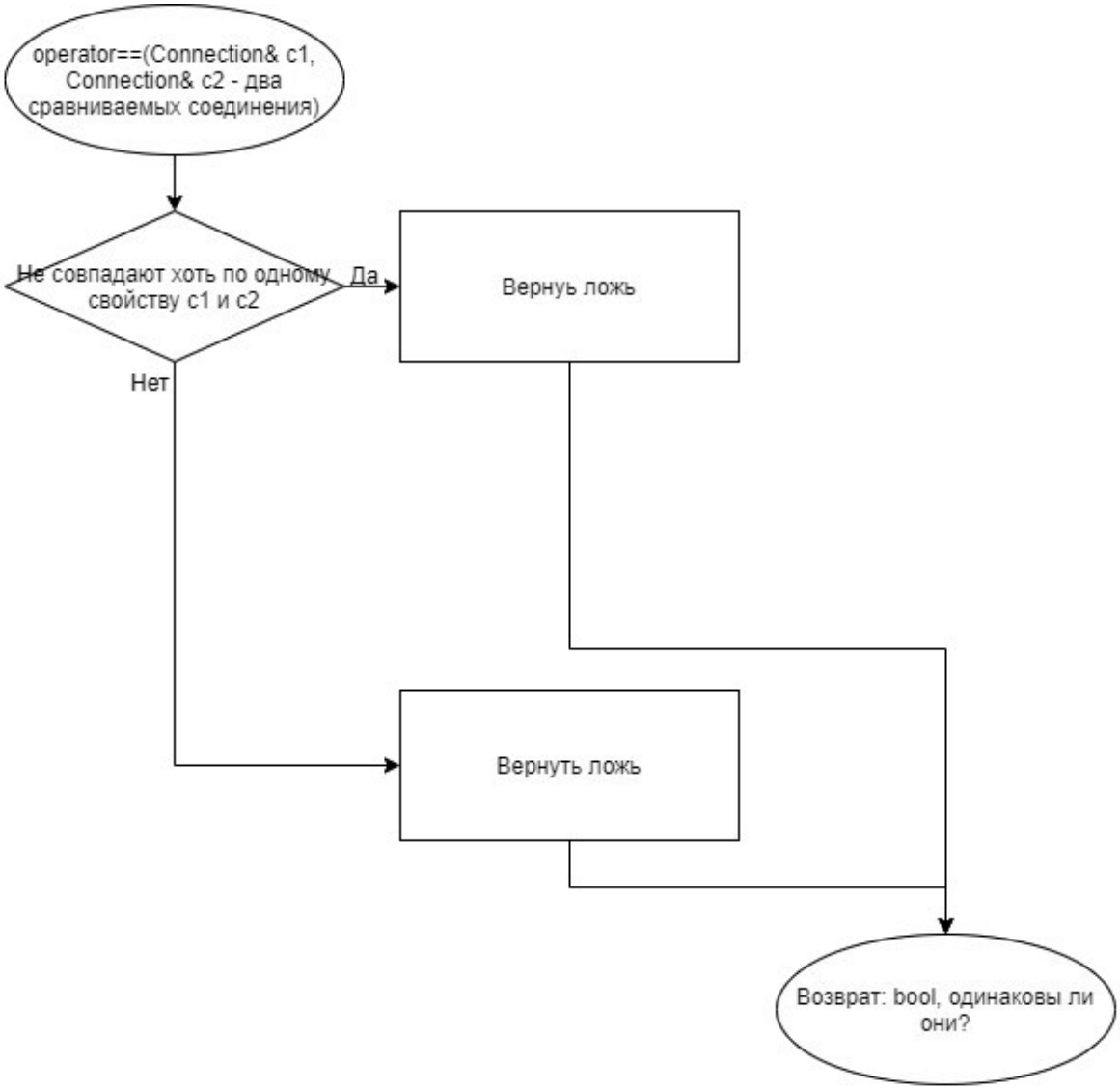
Блок-схема алгоритма

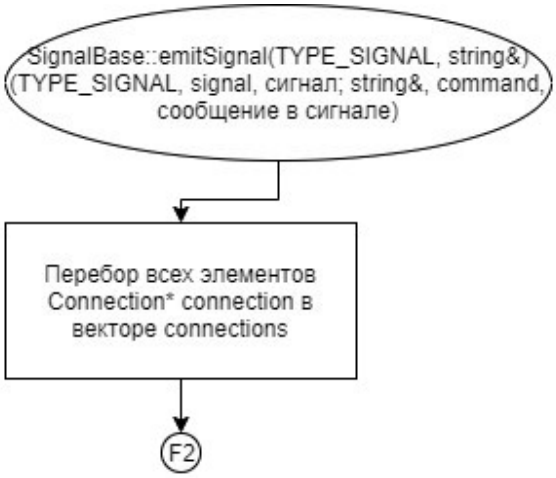

Начало
App::initializeConnections()
Объявление unsigned long id
string emitterName,
handlerName
Инициализация:
Service* emitter = 0, *handler =
0
Создание объекта root класса
App конструктором по
умолчанию
Начало цикла
C3
Вызов метода input
объекта root
A0

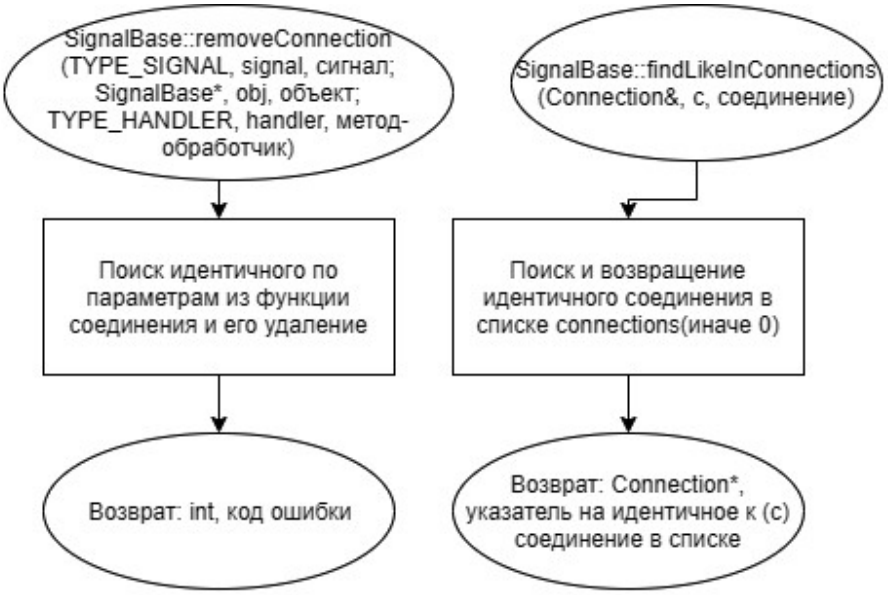

App::initializeSignals()
Ввод сигналов согласно формату, тем самым заполненяя signals до ввода endsignals
Возврат: void
App::getConnectionList()
Перебор всех элементов вектора connectionToPrint и вывод каждого соединения согласно форматированию
Возврат: void

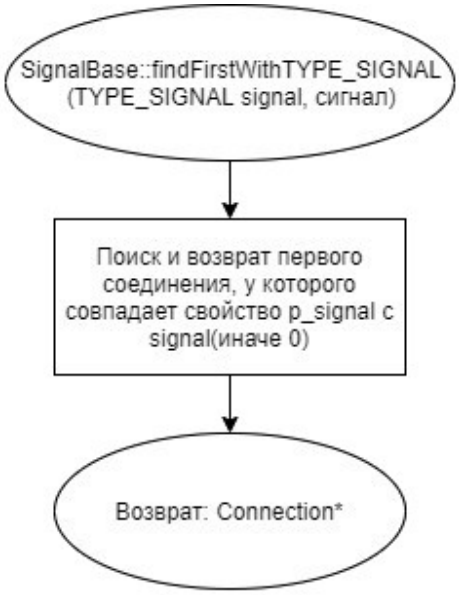

App::getSignalsResult()
signals не пуст
Да
Нет
Вывод "\nEmit signals"
Перебор всех элементов pair<string,string>kv вектора signals
Перебор kv не закончен
Да
Нет
следующий kv
Получение string emitterName и string command из kv
Вызов метода findByAdress(emitterName), положим результат в emitter
A1
Выход из перебора
Возврат: void

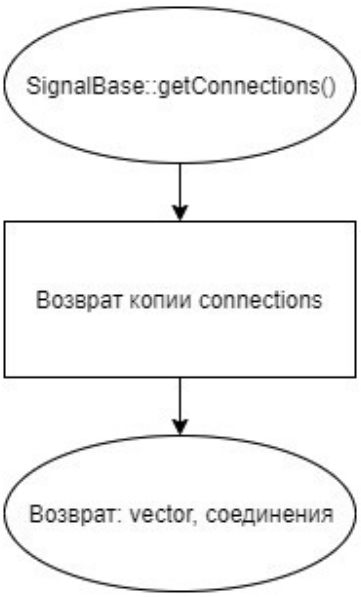

App_Servlet_Asset::handler(string, msg, сообщение)
Получение первого слова из msg и записывание его в emitterName
Стирание первого слова из msg
Получение string handlerName из результата вызова метода getName() этого объекта
Вывод сигнала согласно форматирования
Возврат: void
App_Servlet_Asset::sendSignal (string&, msg, отправляемое сообщение)
Вызов метода emitSignal(преобразованный указатель на метод эмиттера через SIGNAL_D, msg) этого объекта
Возврат: void

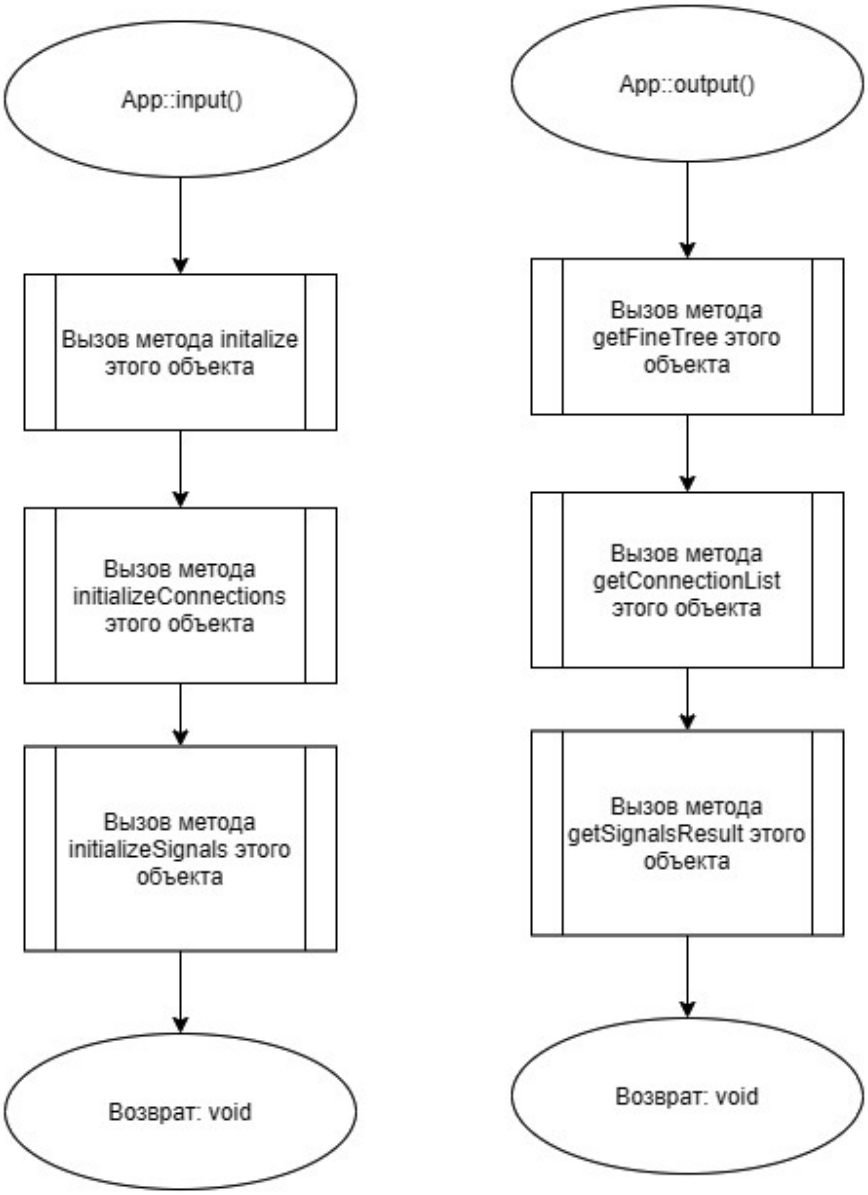

operator==(Connection& c1, Connection& c2 - два сравниваемых соединения)
Не совпадают хоть по одному свойству c1 и c2
Да
Нет
Вернуь ложь
Вернуть ложь
Возврат: bool, одинаковы ли они?

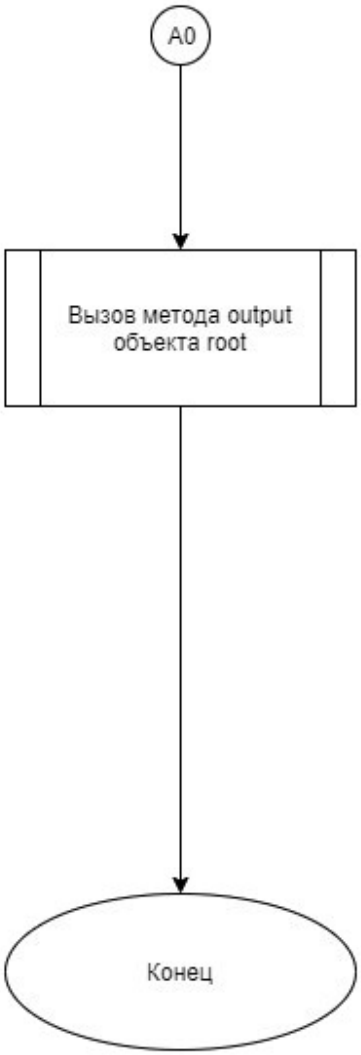

SignalBase::emitSignal(TYPE_SIGNAL, string&)
(TYPE_SIGNAL, signal, сигнал; string&, command, сообщение в сигнале)
Перебор всех элементов Connection* connection в векторе connections
F2

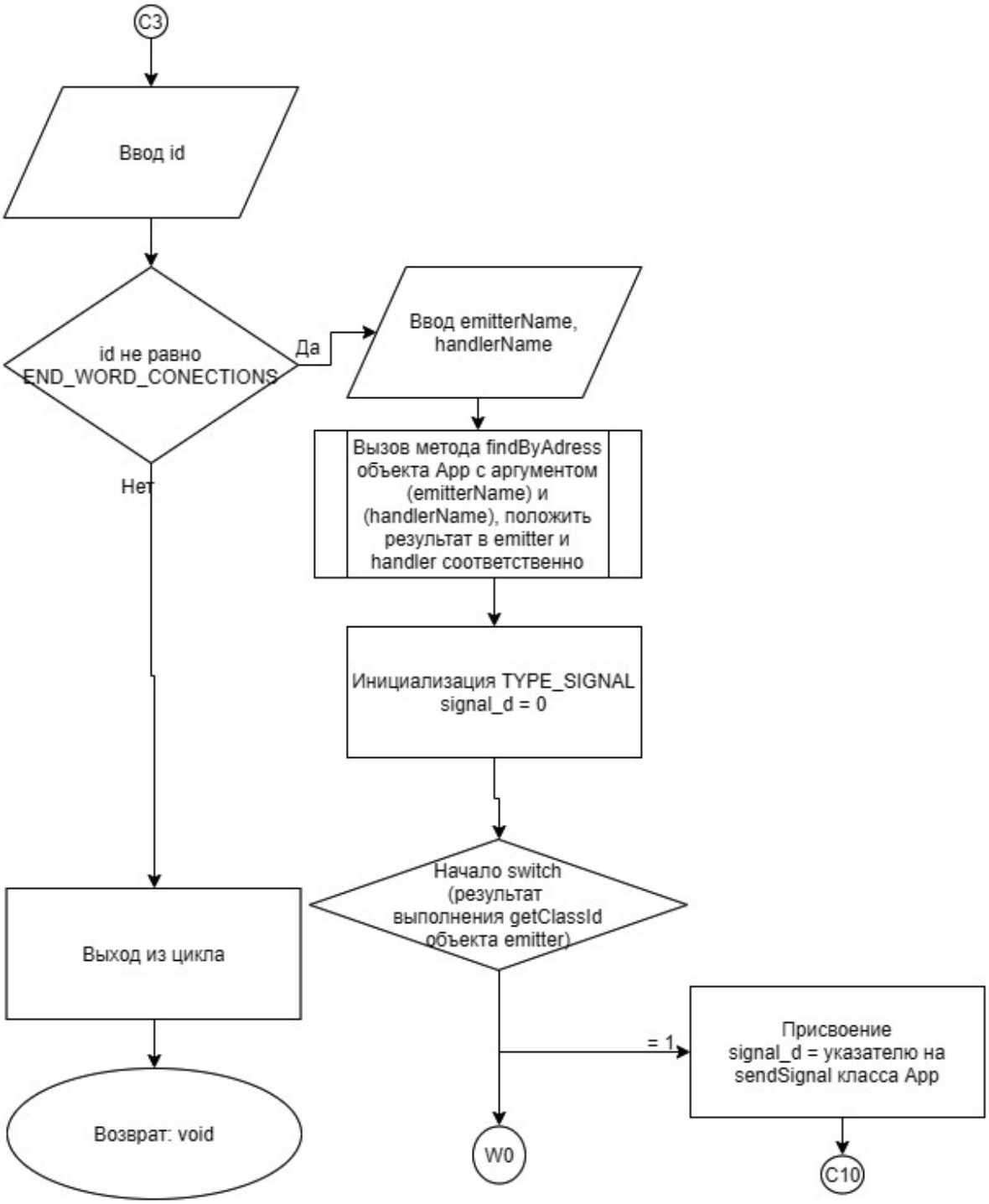

SignalBase::removeConnection (TYPE_SIGNAL, signal, сигнал; SignalBase*, obj, объект; TYPE_HANDLER, handler, метод-обработчик)
Поиск идентичного по параметрам из функции соединения и его удаление
Возврат: int, код ошибки
SignalBase::findLikeInConnections (Connection&, c, соединение)
Поиск и возвращение идентичного соединения в списке connections(иначе 0)
Возврат: Connection*, указатель на идентичное к (c) соединение в списке


SignalBase::findFirstWithTYPE_SIGNAL
(TYPE_SIGNAL signal, сигнал)
Поиск и возврат первого соединения, у которого совпадает свойство p_signal с signal(иначе 0)
Возврат: Connection*

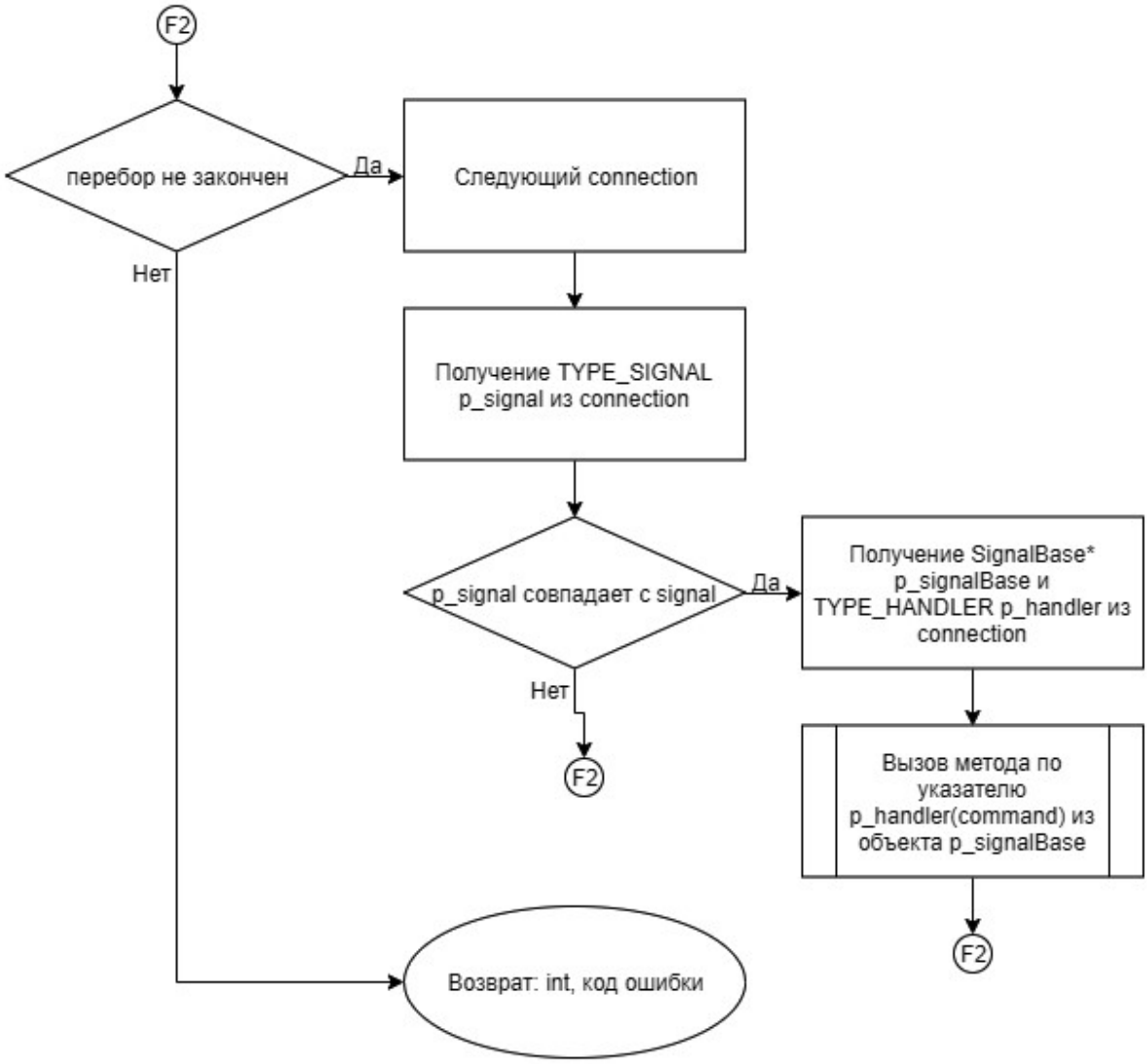

SignalBase::getConnections()
Возврат копии connections
Возврат: vector, соединения

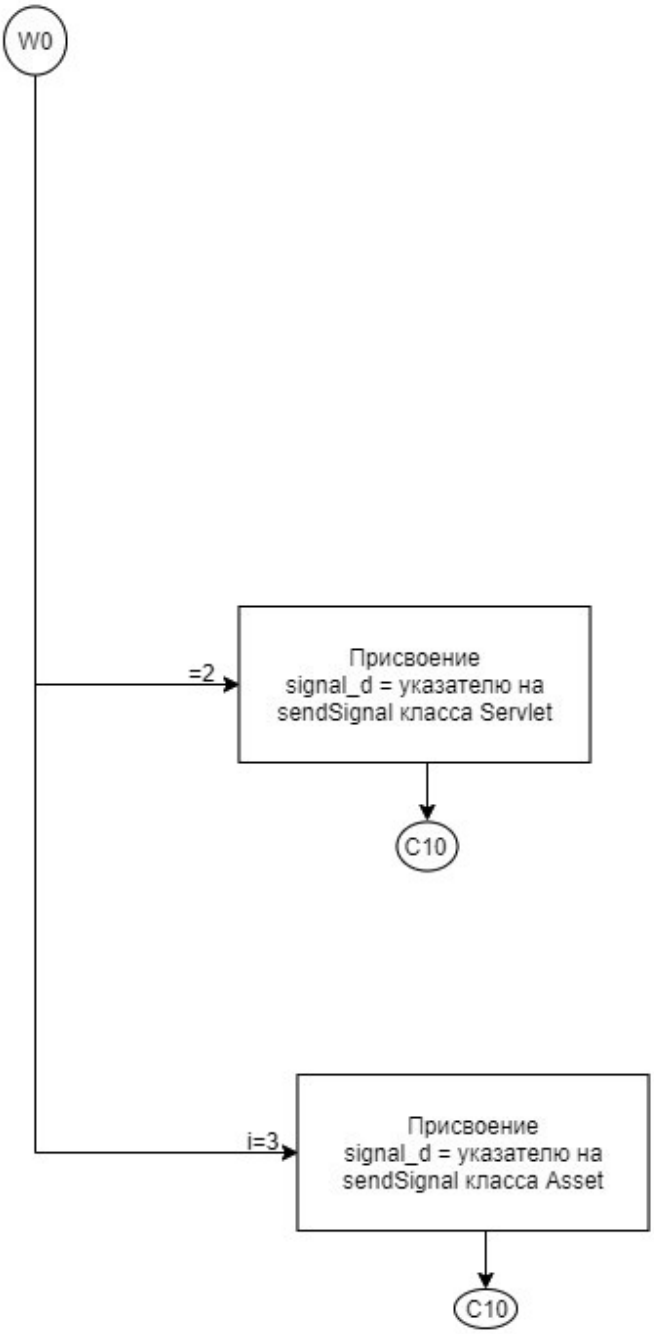

App::input()
Вызов метода initalize этого объекта
Вызов метода initializeConnections этого объекта
Вызов метода initializeSignals этого объекта
Возврат: void
App::output()
Вызов метода getFineTree этого объекта
Вызов метода getConnectionList этого объекта
Вызов метода getSignalsResult этого объекта
Возврат: void

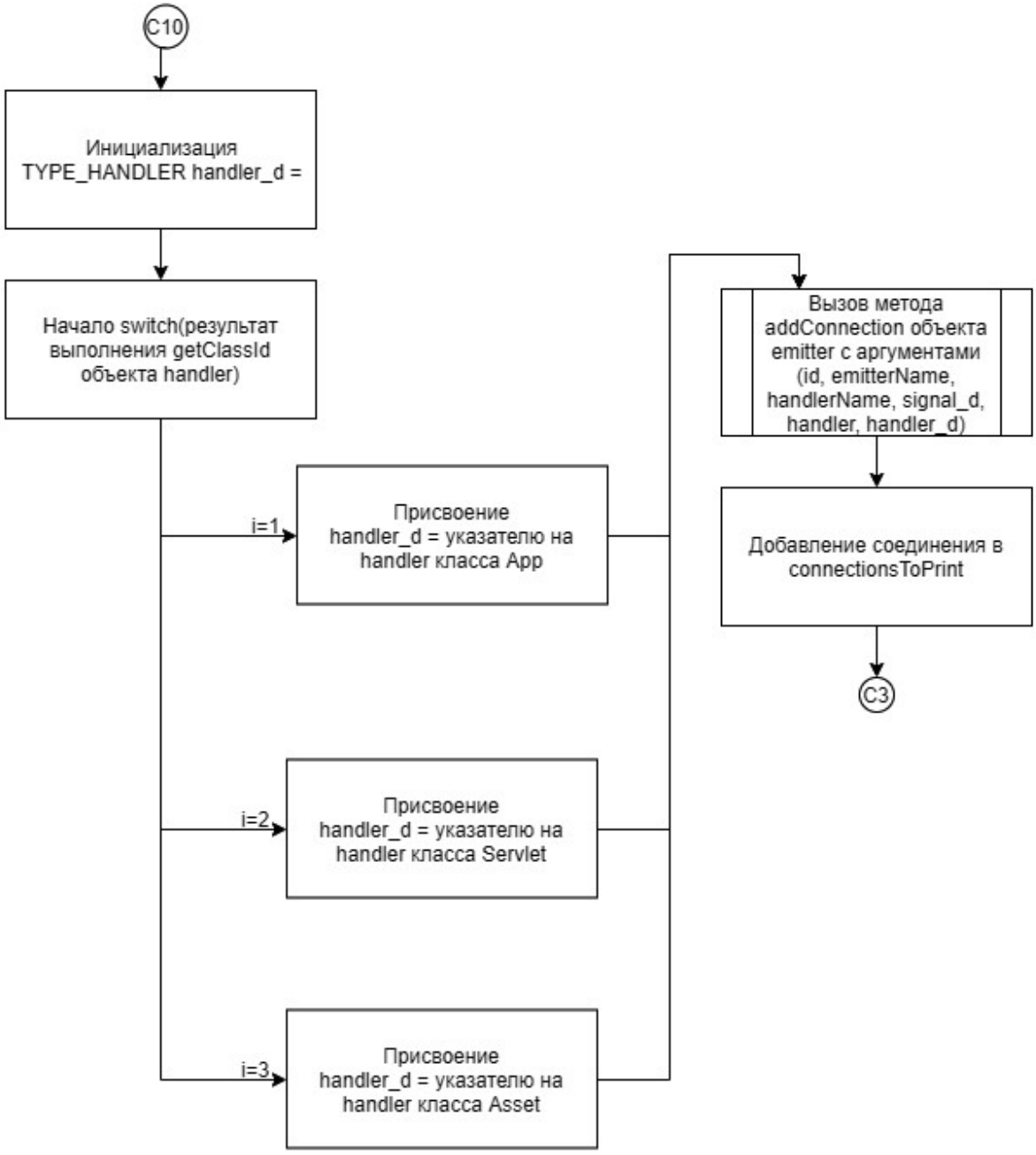

A0
Вызов метода output
объекта root
Конец


C3
Ввод id
id не равно END_WORD_CONECTIONS
Да
Ввод emitterName, handlerName
Вызов метода findByAdress объекта App с аргументом (emitterName) и (handlerName), положить результат в emitter и handler соответственно
Инициализация TYPE_SIGNAL signal_d = 0
Начало switch (результат выполнения getClassId объекта emitter)
= 1
Присвоение signal_d = указателю на sendSignal класса App
C10
W0
Нет
Выход из цикла
Возврат: void


A1
Выбор TYPE_SIGNAL signal_d в зависимости от свойства classId объекта emitter
Добавление в команду имени эмиттера как аргумент
Вызов метода emitSignal(signal_d, command) объекта emitter
D2


F2
перебор не закончен
Да
Следующий connection
Нет
Получение TYPE_SIGNAL p_signal из connection
p_signal совпадает с signal
Да
Получение SignalBase* p_signalBase и TYPE_HANDLER p_handler из connection
Нет
F2
Вызов метода по указателю p_handler(command) из объекта p_signalBase
F2
Возврат: int, код ошибки


W0
=2
Присвоение
signal_d = указателю на
sendSignal класса Servlet
C10
i=3
Присвоение
signal_d = указателю на
sendSignal класса Asset
C10



Код программы

Файл App.cpp

```
#include "App.h"
#include "Servlet.h"
#include "Asset.h"

int App::inputLinesStruct = 1;
int App::inputLinesConnections = 0;
int App::inputLinesSignals = 0;
int App::outputLinesStruct = 0;
int App::outputLinesConnections = 0;
int App::outputLinesSignals = 0;

istream* App::is = &cin;
App::App(): Service("", 1, 1) {
        string name;
        *is >> name;
        App::ss << name << " ";
        setName(name);
}
App::App(const string name) : Service(name, 1, 1) {}

void App::initialize() {
        string p_name, obj_name;
        int classId, status;
        //Ввод состава дерева
        *is >> p_name;
        ss << p_name << " ";
        while (p_name != END_WORD && !(*is).eof() && (*is).good()) {
                *is >> obj_name >> classId >> status;
                ss << obj_name << " ";
                ss << classId << " ";
                ss << status << " ";

                Service* s = 0;
                switch (classId) {
                        case 2:
                        s = (Service*) new Servlet(obj_name, status);
                        break;
                        case 3:
                        s = (Service*) new Asset(obj_name, status);
                        break;
                }

                inputLinesStruct++;
                if (addChild(p_name, s) < 0) {
                        (*is).clear();
                        (*is).ignore(32767, ' ');
                        delete s;
                        cout << "ERR";
                        continue;
                }

                *is >> p_name;
                ss << p_name << " ";
```

```
	}



	/*cout << "The object " << name << " is " << (status > 0 ? "" : "not
") << "ready";
	for (Service *s : children) {
		s->getTree();
	}*/
}

void App::initializeSearch() {
	string adress = "";
	*is >> adress;
	while (adress != END_WORD2 && adress != "") {
		adresses.push_back(adress);
		*is >> adress;
	}
}

void App::getSearchResult() {
	//Координаты искомых объектов
	for (int i = 0; i < (int)adresses.size(); i++) {
		Service* service = 0;
		string adr = adresses[i];
		string name = adr;
		if (name.find("//") == 0) name = name.substr(2);
		if ((int)name.find('/') < 0) {
			service = findByUniqueName(name);
		}
		else {
			service = findByAdress(name);
		}
		cout << endl << adr;
		if (service) cout << " Object name: " << service->getName();
		else cout << " Object not found";
	}
}

void App::initializeConnections() {
	unsigned long id = 0;
	string emitterName, handlerName;
	Service* emitter = 0, * handler = 0;
	istringstream iss;
	while (true) {
		try {
			if (is->eof()) break;
			string line;
			getline(*is, line);
			if (line.empty()) continue;
			iss.str(line);
			string idString;
			iss >> idString;
			id = stoul(idString);
			if (id == END_WORD_CONNECTIONS) break;
			iss >> emitterName >> handlerName;
			inputLinesConnections++;

			emitter = findByAdress(emitterName);
			if (!emitter) throw 5;
```

```
                        handler = findByAdress(handlerName);
                        if (!handler) throw 5;

                        TYPE_SIGNAL signal_d = 0;
                        switch (emitter->getClassId()) {
                                case 1:
                                signal_d = SIGNAL_D(App::sendSignal);
                                break;
                                case 2:
                                signal_d = SIGNAL_D(Servlet::sendSignal);
                                break;
                                case 3:
                                signal_d = SIGNAL_D(Asset::sendSignal);
                                break;
                                default: throw 5;
                        }

                        TYPE_HANDLER handler_d = 0;
                        switch (handler->getClassId()) {
                                case 1:
                                handler_d = HANDLER_D(App::handler);
                                break;
                                case 2:
                                handler_d = HANDLER_D(Servlet::handler);
                                break;
                                case 3:
                                handler_d = HANDLER_D(Asset::handler);
                                break;
                                default: throw 5;
                        }

                        if (emitter->addConnection(id, emitterName,
handlerName, handlerName, signal_d, handler, handler_d) < 0)
inputLinesConnections--;
                        connectionsToPrint.push_back(make_pair(id,
make_pair(emitterName, handlerName)));
                } catch (...) {
                        cout << "ERRIConnections(inputLinesConnections=" <<
inputLinesConnections << ")";
                }
                iss.clear();
        }
}

void App::initializeSignals() {
        string emitterName, text;
        (*is).clear();
        (*is) >> emitterName;
        while (emitterName != END_WORD_SIGNALS && emitterName != "" && !
(*is).eof()) {
                (*is) >> text;
                (*is).ignore(32767, '\n');
                inputLinesSignals++;

                signals.push_back(make_pair(emitterName, text));
                emitterName = "";
```

```
                (*is) >> emitterName;
        }
}

void App::getConnectionList() {
        vector<Connection*> connections = SignalBase::getConnections();
        //printConnections();
        if (connectionsToPrint.size() > 0) cout << "\nSet connects";
        for (auto p : connectionsToPrint) {
                cout << "\n" << p.first << " " << p.second.first << " " <<
p.second.second;
        }
        for (; outputLinesConnections < inputLinesConnections;
outputLinesConnections++) {
                //cout << "\nFILL CONNECTIONS";
        }
}

void App::getSignalsResult() {
        if (!signals.empty()) cout << "\nEmit signals";
        for (pair<string, string> kv : signals) {
                string emitterName = kv.first;
                string command = kv.second;
                outputLinesSignals++;
                int emitterClassId = 0;

                try {
                        Service* emitter = findByAdress(emitterName);
                        if (!emitter) throw -1;

                        TYPE_SIGNAL signal_d = 0;
                        emitterClassId = emitter->getClassId();
                        switch (emitterClassId) {
                                case 1:
                                signal_d = SIGNAL_D(App::sendSignal);
                                break;
                                case 2:
                                signal_d = SIGNAL_D(Servlet::sendSignal);
                                break;
                                case 3:
                                signal_d = SIGNAL_D(Asset::sendSignal);
                                break;
                                default: throw -2;
                        }

                        command = emitter->getName() + " " + command;
                        emitter->emitSignal(signal_d, command);
                } catch (int e) {
                        cout << "ERROSignals(";
                        switch (e) {
                                case -1:
                                cout << "There is no object with name \"" <<
emitterName << "\" among object tree";
                                break;
                                case -2:
                                printf("There is no type \"%d\"",
emitterClassId);
                                break;
                        }
                        cout << ")";
```

```
                }

        }
        for (; outputLinesSignals < inputLinesSignals; outputLinesSignals++) {
                cout << "\nFILL CONNECTIONS";
        }
}

void App::input() {
        initialize();
        initializeConnections();
        initializeSignals();
        //dynamic_cast<ifstream*>(App::is)->close();
}

void App::output() {
        getFineTree();
        getConnectionList();
        getSignalsResult();
}

void App::handler(string msg) {
        stringstream ss = stringstream(msg);
        string emitterName;
        ss >> emitterName;
        msg = msg.erase(0, emitterName.length() + 1);
        string handlerName = getName();
        cout << "\nSignal to " << handlerName << " Text: " << emitterName << "
-> ";
        cout << msg;
}

void App::sendSignal(string& msg) {
        SignalBase::emitSignal(SIGNAL_D(App::sendSignal), msg);
}
```

Файл App.h

```
#ifndef APP_H
#define APP_H

#include "Service.h"
#include <set>
#include <sstream>
#include <fstream>

//Root
class App : public Service {
private:
        const string END_WORD = "endtree";
```

```
        const string END_WORD2 = "//";
        const char END_WORD_CONNECTIONS = 0;
        const string END_WORD_SIGNALS = "endsignals";
        vector<string> adresses;
        vector<pair<string, string>> signals;
        vector<pair<int, pair<string,string>>> connectionsToPrint;
public:
        static stringstream ss;
        static int inputLinesStruct;
        static int inputLinesConnections;
        static int inputLinesSignals;
        static int outputLinesStruct;
        static int outputLinesConnections;
        static int outputLinesSignals;
        static istream* is;
public:
        App();
        App(const string name);
        void initialize();
        void initializeSearch();
        void initializeConnections();
        void initializeSignals();
        void getSearchResult();
        void getConnectionList();
        void getSignalsResult();
        void input();
        void output();
        void handler(string msg);
        void sendSignal(string& msg);
};

#endif
```

Файл Asset.cpp

```
#include "Asset.h"

Asset::Asset(const string name, const int status, const int size) :
Service(name, 3, status) {
        this->size = size;
}

int Asset::getSize() {
        return size;
}

void Asset::handler(string msg) {
        stringstream ss = stringstream(msg);
        string emitterName;
        ss >> emitterName;
        msg = msg.erase(0, emitterName.length() + 1);
        string handlerName = getName();
        cout << "\nSignal to " << handlerName << " Text: " << emitterName << "
-> ";
```

```
        cout << msg;
}

void Asset::sendSignal(string& msg) {
        SignalBase::emitSignal(SIGNAL_D(Asset::sendSignal), msg);
}
```

Файл Asset.h

```
#ifndef ASSET_H
#define ASSET_H

#include "Service.h"

class Asset : public Service {
private:
        int size = 0;
public:
        Asset(const string name, const int status, const int size = 0);
        int getSize();
        void handler(string msg);
        void sendSignal(string& msg);
};

#endif
```

Файл main.cpp

```
#include "main.h"


int main() {
        App root = App();
        root.input();
        root.output();

        //system("pause");
        return(0);
}
```

Файл main.h

```
#ifndef MAIN_H
#define MAIN_H

#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <iostream>
#include <fstream>
#include "App.h"

using namespace std;

#endif
```

Файл ServiceChilds.cpp

```
/*#include "ServiceChilds.h"

App::App() : Service() {}

App::App(Service* head, string name, int status) : Service(head, name, status)
{
        cin >> this->name;
}

Servlet::Servlet() : Service() {}

Servlet::Servlet(Service* head, string name, int status) : Service(head, name,
status) {}

Asset::Asset() : Service() {}

Asset::Asset(Service* head, string name, int status) : Service(head, name,
status) {}
*/
```

Файл ServiceChilds.h

```
#ifndef _1_H
#define _1_H
/*
#include "Service.h"
```

```
class App : public Service {
public:
	App();
	App(Service* head, string name, int status);
};
class Servlet : public Service {
public:
	Servlet();
	Servlet(Service* head, string name, int status);
};

class Asset : public Service {
public:
	Asset();
	Asset(Service* head, string name, int status);
};
*/
#endif
```

Файл Service.cpp

```
#include "Service.h"
#include "App.h"

unsigned long Service::UUID_seq = 1;
stringstream App::ss = stringstream("");

bool Service::isSetConnectsPrinted = false;

Service::Service(string name, int classId, int status) {
	UUID = UUID_seq++;
	this->name = name;
	this->classId = classId;
	this->status = status;
}

int Service::addChild(string parentName, Service* service) {
	if (!service) return -1;
	if (parentName.find("//") == 0) parentName = parentName.substr(2);
	Service* parent = 0;
	if ((int)parentName.find('/') < 0) {
		parent = findByUniqueName(parentName);
	} else {
		parent = findByAdress(parentName);
	}
	if (parent) {
		parent->children.push_back(service);
		return 0;
	}
	return -1;
```

```
}

Service* Service::findByUniqueName(const string name, int steps) {
	if (this->name == name) return this;
	if (steps == 0) return 0;
	for (Service* s : children) {
		if (s->name == name) return s;
		Service* sr = s->findByUniqueName(name, steps - 1);
		if (sr) return sr;
	}
	return NULL;
}

void Service::getTree() {
	cout << "Object tree";
	cout << "\n";
	cout << "The object " << name << " is " << (status > 0 ? "" : "not ")
<< "ready";
	for (int i = 0; i < (int)children.size(); i++) {
		children[i]->getTree();
	}
}

void Service::getFineTree(int level) {
	if (level == 0) cout << "Object tree";
	if (false && level == 3 && name == "object_09") {
		cout << "[";
		cout << App::ss.str();
		cout << "]\nHAHAHAH9";
	} else {
		App::outputLinesStruct++;
		cout << "\n";
		for (int i = 0; i < level; i++) {
			cout << TAB;
		}
		cout << name;
	}
	for (int i = 0; i < (int)children.size(); i++) {
		children[i]->getFineTree(level + 1);
	}
	if (level == 0) {
		for (; App::outputLinesStruct < App::inputLinesStruct;
App::outputLinesStruct++) cout << "\nFILL STRUCT";
	}
}

unsigned long Service::getUUID() {
	return UUID;
}

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

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

int Service::getClassId() {
	return classId;
```

```
}

int Service::getStatus() {
	return status;
}

string Service::getPathItem(string adress, int level) {
	if (adress[0] != '/') adress = '/'+adress;
	int item_start = 1, item_end;
	int currentLevel = 1;
	while (item_start) {
		item_end = adress.find('/', item_start);
		if (currentLevel == level) return adress.substr(item_start,
item_end - item_start);
		currentLevel++;
		item_start = item_end + 1;
	}
	return "";
}

Service* Service::findByAdress(string adress) {
	//Проверка, что это адрес
	if (adress.find("//") == 0) adress = adress.substr(2);
	if ((int)adress.find('/') < 0) {
		return findByUniqueName(adress);
	}
	//


	int level = 1;
	string pathItem = getPathItem(adress, level);
	level++;
	if (pathItem != name) return 0;
	Service* service = this;
	while (true) {
		pathItem = getPathItem(adress, level);
		level++;
		if (pathItem.empty()) return service;
		Service* child = service->findByUniqueName(pathItem, 1);
		if (!child) return 0;
		service = child;
	}
	return 0;
}

void Service::printConnections() {
	vector<Connection*> connections = getConnections();
	for (Connection* c : connections) {
		App::outputLinesConnections++;
		if (!isSetConnectsPrinted) {
			isSetConnectsPrinted = true;
			cout << "\nSet connects";
		}
		string emitterAdress = c->emitterAdress, handlerName =
"Untitled";
		Service* handler = dynamic_cast<Service*>(c->p_signalBase);
```

```
                if (handler) handlerName = handler->getName();
                string handlerAdress = c->handlerAdress;
                cout << "\n" << c->id << " " << emitterAdress << " " <<
handlerAdress;
        }
        for (auto child : children) {
                child->printConnections();
        }
}
```

Файл Service.h

```
#ifndef SERVICE_H
#define SERVICE_H

#include <string>
#include <vector>
#include <iostream>
#include <sstream>
#include <map>
#include "SignalBase.h"

using namespace std;

class Service : public SignalBase {
private:
        static bool isSetConnectsPrinted;
        static unsigned long UUID_seq;
        unsigned long UUID;
        string name = "";
        int classId = 1;
        int status = 0;
        vector<Service*> children;
public:
        const string TAB = "    ";
private:
        static string getPathItem(string, int);
protected:
public:
        Service(const string name, const int classId, const int status);
        int addChild(const string parentName, Service* service);
        Service* findByUniqueName(const string name, int steps = -1);
        void getTree();
        void getFineTree(int level = 0);
        Service* findByAdress(const string);
public:
        unsigned long getUUID();
        string getName();
        void setName(string name);
        int getClassId();
        int getStatus();
        void printConnections();
};
```

```
#endif
```

Файл Servlet.cpp

```
#include "Servlet.h"

Servlet::Servlet(const string name, const int status, const string serverIp) :
Service(name, 2, status) {
	this->serverIp = serverIp;
}

string Servlet::getServerIp() {
	return serverIp;
}

void Servlet::handler(string msg) {
	stringstream ss = stringstream(msg);
	string emitterName;
	ss >> emitterName;
	msg = msg.erase(0, emitterName.length() + 1);
	string handlerName = getName();
	cout << "\nSignal to " << handlerName << " Text: " << emitterName << "
-> ";
	cout << msg;
}

void Servlet::sendSignal(string& msg) {
	SignalBase::emitSignal(SIGNAL_D(Servlet::sendSignal), msg);
}
```

Файл Servlet.h

```
#ifndef SERVLET_H
#define SERVLET_H

#include "Service.h"

class Servlet : public Service {
private:
	string serverIp = "";
public:
	Servlet(const string name, const int status, const string serverIp =
"");
	string getServerIp();
```

```
        void handler(string msg);
        void sendSignal(string& msg);
};

#endif
```

Файл SignalBase.cpp

```
#include "SignalBase.h"

int SignalBase::addConnection(unsigned long id, string emitterAdress, string
handlerAdress, string handlerName, TYPE_SIGNAL signal, SignalBase* obj,
TYPE_HANDLER handler) {

        Connection* c;
        c = new Connection();
        c->id = id;
        c->emitterAdress = emitterAdress;
        c->handlerAdress = handlerAdress;
        c->p_signal = signal;
        c->p_signalBase = obj;
        c->p_handler = handler;

        if (findLikeInConnections(*c)) {
                delete c;
                return -1;
        }
        connections.push_back(c);
        return 0;
}

int SignalBase::removeConnection(TYPE_SIGNAL signal, SignalBase* obj,
TYPE_HANDLER handler) {
        Connection c;
        c.p_signal = signal;
        c.p_signalBase = obj;
        c.p_handler = handler;
        Connection* connectionToDelete = findLikeInConnections(c);
        if (!connectionToDelete) return -1;

        vector<Connection*>::iterator it = find(connections.begin(),
connections.end(), connectionToDelete);

        connections.erase(it);
        return 0;
}
int SignalBase::emitSignal(TYPE_SIGNAL signal, string& command) {
        for (Connection* connection : connections) {
                TYPE_SIGNAL p_signal = connection->p_signal;
                if (p_signal == signal) {
                        SignalBase* p_signalBase = connection->p_signalBase;
                        TYPE_HANDLER p_handler = connection->p_handler;
                        (p_signalBase->*p_handler)(command);
                }
```

```
        }
        return 0;
}

Connection* SignalBase::findLikeInConnections(Connection &c) {
        Connection* it;
        for (int i = 0; i < connections.size(); i++) {
                it = connections[i];
                if (*it == c) {
                        return it;
                }
        }
        return 0;
}

vector<Connection*> SignalBase::getConnections() {
        return vector<Connection*>(connections);
}

Connection* SignalBase::findFirstWithTYPE_SIGNAL(TYPE_SIGNAL signal) {
        for (Connection* c : connections) {
                if (c->p_signal == signal) return c;
        }
        return 0;
}

bool operator==(Connection& c1, Connection& c2)
{
        if (c1.id != c2.id) return false;
        if (c1.p_signal != c2.p_signal) return false;
        if (c1.p_signalBase != c2.p_signalBase) return false;
        if (c1.p_handler != c2.p_handler) return false;
        return true;
}
```

Файл SignalBase.h

```
#ifndef SIGNALBASE_H
#define SIGNALBASE_H

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

using namespace std;

class SignalBase;

typedef void (SignalBase ::* TYPE_SIGNAL) (string&);
```

```
typedef void (SignalBase ::* TYPE_HANDLER) (string);

#define SIGNAL_D(signal_f) (TYPE_SIGNAL) (&signal_f)
#define HANDLER_D(handler_f) (TYPE_HANDLER) (&handler_f)

struct Connection {
        unsigned long id;
        TYPE_SIGNAL p_signal;
        SignalBase* p_signalBase;
        TYPE_HANDLER p_handler;
        string emitterAdress;
        string handlerAdress;
        friend bool operator==(Connection&, Connection&);
};

class SignalBase {
private:
        vector<Connection*> connections;
protected:
public:
        int emitSignal(TYPE_SIGNAL, string&);
        int addConnection(unsigned long, string, string, string, TYPE_SIGNAL,
SignalBase*, TYPE_HANDLER);
        int removeConnection(TYPE_SIGNAL, SignalBase*, TYPE_HANDLER);
        Connection* findLikeInConnections(Connection& c);
        Connection* findFirstWithTYPE_SIGNAL(TYPE_SIGNAL);
        virtual vector<Connection*> getConnections();
};

#endif
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root root ob_1 3 1 ob_1 ob_2 2 1 root ob_3 3 -1 ob_3 ob_4 3 1 ob_4 ob_5 2 1 ob_3 ob_6 3 1 ob_6 ob_7 2 1 endtree 0 endsignals	Object tree root ob_1 ob_2 ob_3 ob_4 ob_5 ob_6 ob_7	Object tree root ob_1 ob_2 ob_3 ob_4 ob_5 ob_6 ob_7
a a a_1 2 2 a c 2 2 endtree 1 a c	Object tree a a_1 c Set connects	Object tree a a_1 c Set

0 a hi! endsignals	1 a c Emit signals Signal to c Text: a -> hi!	connects 1 a c Emit signals Signal to c Text: a -> hi!
b b b_2 2 2 b_2 c 2 0 b a_1 2 -1 endtree 1 b c 2 a_1 b 0 b hi! a_1 bye! endsignals	Object tree b b_2 c a_1 Set connects 1 b c 2 a_1 b Emit signals Signal to c Text: b -> hi! Signal to b Text: a_1 -> bye!	Object tree b b_2 c a_1 Set connects 1 b c 2 a_1 b Emit signals Signal to c Text: b -> hi! Signal to b Text: a_1 -> bye!
root root a 2 1 endtree 1 root a 2 a root 3 root a 0 a 123 a 321 root abc endsignals	Object tree root a Set connects 1 root a 2 a root 3 root a Emit signals Signal to root Text: a -> 123 Signal to root Text: a -> 321 Signal to a Text: root -> abc Signal to a Text: root -> abc	Object tree root a Set connects 1 root a 2 a root 3 root a Emit signals Signal to root Text: a -> 123 Signal to root Text: a -> 321 Signal to a Text: root -> abc Signal to a Text: root -> abc
root root a 2 1 endtree 1 root a 2 a root 0 a 123 a 321 root abc endsignals	Object tree root a Set connects 1 root a 2 a root Emit signals Signal to root Text: a -> 123 Signal to root Text: a -> 321 Signal to a Text: root -> abc	Object tree root a Set connects 1 root a 2 a root Emit signals Signal to root Text: a -> 123 Signal to root Text: a -> 321 Signal to a Text: root -> abc
r /r o1 2 2 /r o2 2 2 /r o3 2 2 /r o4 2 2 /r o5 2 2 /r o6 2 2 /r o7 2 2 /r/o4 o5 2 2 /r/o4/o5 o5 2 2 endtree 1 /r/o4/o5 o6 0 /r/o4/o5 123 endsignals	Object tree r o1 o2 o3 o4 o5 o5 o5 o6 o7 Set connects 1 /r/o4/o5 o6 Emit signals Signal to o6 Text: o5 -> 123	Object tree r o1 o2 o3 o4 o5 o5 o5 o6 o7 Set connects 1 /r/o4/o5 o6 Emit signals Signal to o6 Text: o5 -> 123
root /root object_1 3 1 /root object_2 2 1 /root/object_2 object_4 3 -1 /root/object_2 object_5 2 1 /root object_3 3 1 /root/object_2 object_3 2 1 /root/object_1 object_7 3 1 /root/object_2/object_4 object_7 3 -1 endtree 0 endsignals	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3

app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 object_1 object_7 2 1 endtree 0 endsignals 1 object_7 object_1 2 object_7 object_2 3 object_7 object_3 4 object_7 object_7 5 object_7 object_7 0 object_7 1a endsignals	Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3	Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3
app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 //object_1 object_7 2 1 endtree 0 endsignals	Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3	Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3
appli_root /appli_root object_01 3 1 /appli_root object_02 2 1 /appli_root/object_02 object_04 3 -1 /appli_root/object_02 object_05 2 1 /appli_root object_04 3 1 /appli_root/object_02 object_06 2 3 /appli_root/object_02/object_06 object_08 2 4 /appli_root/object_04 object_09 2 5 /appli_root/object_01 object_07 2 1 endtree 0 endsignals	Object tree appli_root object_01 object_07 object_02 object_04 object_05 object_06 object_08 object_04 object_09	Object tree appli_root object_01 object_07 object_02 object_04 object_05 object_06 object_08 object_04 object_09