Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

Объектно-ориентированное программирование (ООП) сегодня трудно переоценить. В современном мире, где программное обеспечение пронизывает практически все сферы нашей жизни, способность создавать эффективные, гибкие и легко поддерживаемые программы становится ключевым навыком.

ООП предлагает мощный набор инструментов для разработки сложных программных систем, предоставляя четкую структуру для организации кода и управления его сложностью. Одним из основных принципов ООП является инкапсуляция, которая позволяет скрывать детали реализации и предоставлять только необходимый интерфейс для работы с объектами.

Наследование и полиморфизм также являются важными понятиями в ООП, позволяя переиспользовать код и создавать более гибкие и масштабируемые программы. Понимание принципов ООП является необходимым условием для успешной карьеры в сфере программирования.

Оно позволяет создавать более качественное программное обеспечение, сокращать время разработки и оптимизировать процесс работы. Таким образом, ООП является основой современной разработки программного обеспечения и становится все более важным инструментом для программистов во всех областях индустрии информационных технологий.

1 ПОСТАНОВКА ЗАДАЧИ

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

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

- установления связи между сигналом текущего объекта и обработчиком целевого объекта;
- удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;
- выдачи сигнала от текущего объекта с передачей строковой переменной. Включенный объект может выдать или обработать сигнал.

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

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

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

1. Если текущий объект отключен, то выход, иначе к пункту 2.
2. Вызов метода сигнала с передачей строковой переменной по ссылке.
3. Цикл по всем связям сигнал-обработчик текущего объекта:

    3.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то проверить готовность целевого объекта. Если целевой объект готов, то вызвать метод обработчика

целевого объекта указанной в связи и передать в качестве аргумента строковую переменную по значению.

4. Конец цикла.

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

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

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 3 курсовой работы. Если при построении дерева иерархии возникает ситуация дубляжа имен среди починенных у текущего головного объекта, то новый объект не создается.

Система содержит объекты шести классов с номерами: 1, 2, 3, 4, 5, 6. Классу корневого объекта соответствует номер 1. В каждом производном классе реализовать один метод сигнала и один метод обработчика.

Каждый метод сигнала с новой строки выводит:

```
Signal from «абсолютная координата объекта»
```

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

```
«пробел»(class: «номер класса»)
```

Каждый метод обработчика с новой строки выводит:

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

Моделировать работу системы, которая выполняет следующие команды с параметрами:

- EMIT «координата объекта» «текст» – выдает сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата

целевого объекта» – устанавливает связь;

- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаляет связь;
- SET_CONDITION «координата объекта» «значение состояния» – устанавливает состояние объекта.
- END – завершает функционирование системы (выполнение программы).

Реализовать алгоритм работы системы:

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

Допускаем, что все входные данные вводятся синтаксически корректно. Контроль корректности входных данных можно реализовать для самоконтроля работы программы. Не оговоренные, но необходимые функции и элементы классов добавляются разработчиком.

1.1 Описание входных данных

В методе построения системы.

Множество объектов, их характеристики и расположение на дереве

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

После ввода состава дерева иерархии построчно вводится:

```
«координата объекта выдающего сигнал» «координата целевого объекта»
```

Ввод информации для построения связей завершается строкой, которая содержит:

```
«end_of_connections»
```

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

- EMIT «координата объекта» «текст» – выдать сигнал от заданного по координате объекта;
- SET_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – установка связи;
- DELETE_CONNECT «координата объекта выдающего сигнал» «координата целевого объекта» – удаление связи;
- SET_CONDITION «координата объекта» «значение состояния» – установка состояния объекта.
- END – завершить функционирование системы (выполнение программы).

Команда END присутствует обязательно.

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

Если не найден объект по координате:

```
Object «координата объекта» not found
```

Если не найден целевой объект по координате:

```
Handler object «координата целевого объекта» not found
```

Пример ввода:

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

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева.

Далее, построчно, если отработал метод сигнала:

```
Signal from «абсолютная координата объекта»
```

Если отработал метод обработчика:

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

Пример вывода:

```
Object tree
appls_root
    object_s1
        object_s7
    object_s2
        object_s4
        object_s6
    object_s13
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 1 (class: 4)
Signal to / Text:  Send message 1 (class: 4)
Signal from /object_s2/object_s4
```

```
Signal to /object_s2/object_s6 Text:  Send message 2 (class: 4)
Signal to / Text:  Send message 2 (class: 4)
Signal from /object_s2/object_s4
Signal to /object_s2/object_s6 Text:  Send message 3 (class: 4)
Signal to / Text:  Send message 3 (class: 4)
Signal from /object_s1
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект ob_cl_application класса cl_application предназначен для для заупска прилоения;
- функция main для основная.

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм конструктора класса cl_application

Функционал: конструктор.

Параметры: нет.

Алгоритм конструктора представлен в таблице 1.

Таблица 1 – Алгоритм конструктора класса cl_application

№	Предикат	Действия	№ перехода
1		p_head_obj =head	∅

3.2 Алгоритм метода singnal_f класса cl_application

Функционал: для работы с сигналами.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода singnal_f класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + абсолютный путь message "class: " + номер класса	∅

3.3 Алгоритм метода handler_f класса cl_application

Функционал: для работы с сигналамии.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода handler_f класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Signal from " + результат поиска абсолютного пути	∅

3.4 Алгоритм метода get_class_num класса cl_application

Функционал: получает номер класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_class_num класса cl_application

№	Предикат	Действия	№ перехода
1		возвращает номер класса	∅

3.5 Алгоритм метода get_class класса cl_base

Функционал: получение номера класса.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_class класса cl_base

№	Предикат	Действия	№ перехода
1		возвращает номер класса	∅

3.6 Алгоритм метода set_connection класса cl_base

Функционал: установить соединение.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода set_connection класса cl_base

№	Предикат	Действия	№ перехода
1		Объявляем объект p_value класса o_sh Инициализируем i=0	2
2	i<размер connects		3
			4
3	Все параметры равны значениям свойств connects от i	break	∅
		i+=1	4
4		Приравниваем все свойства p_value к параметрам дОбавляем p_value в список connects	∅

3.7 Алгоритм метода delete_connection класса cl_base

Функционал: удалить соединение.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода delete_connection класса cl_base

№	Предикат	Действия	№ перехода
1		Объявляем список p_it типа o_sh* p_it первый объект в списке connects	2
2	p_it = последний объект в списке connects		∅
			3
3	Все параметры равны значениям свойств p_it	удалить p_it в списке connects	∅
		p_it = следующий объект в списке connection	4
4		elfkbnm p_it	5
5		p_it предыдущий элемент в списке connects	6
6		Инициализация connect как первый элемент в списке connects	∅

3.8 Алгоритм метода get_abs_path класса cl_base

Функционал: абсолютный путь.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get_abs_path класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация строки path= "" 2 2 Инициализцаия p_object класса cl_base* равному этому объекту	2
2	Головной объект p_obj не равен nullptr	path = "/" + имя p_object + path	3
			3

№	Предикат	Действия	№ перехода
3		p_object равен своему головному объекту 3 5 Вернуть path	4
4		вернуть path	∅

3.9 Алгоритм функции main

Функционал: основная функция.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 9.

Таблица 9 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта класса cl_application 2 2 Вызов метода build_tree_object 3 3 Вызов метода exec_app	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-6.

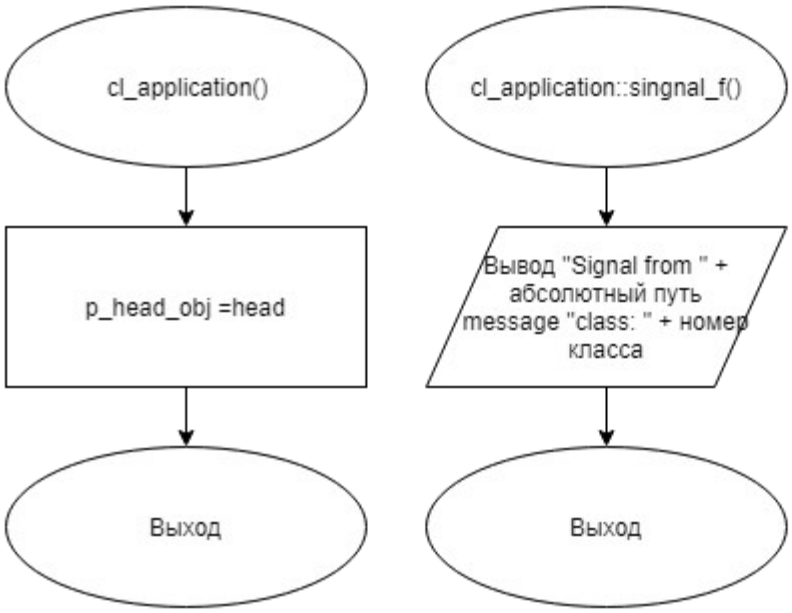

Рисунок 1 – Блок-схема алгоритма

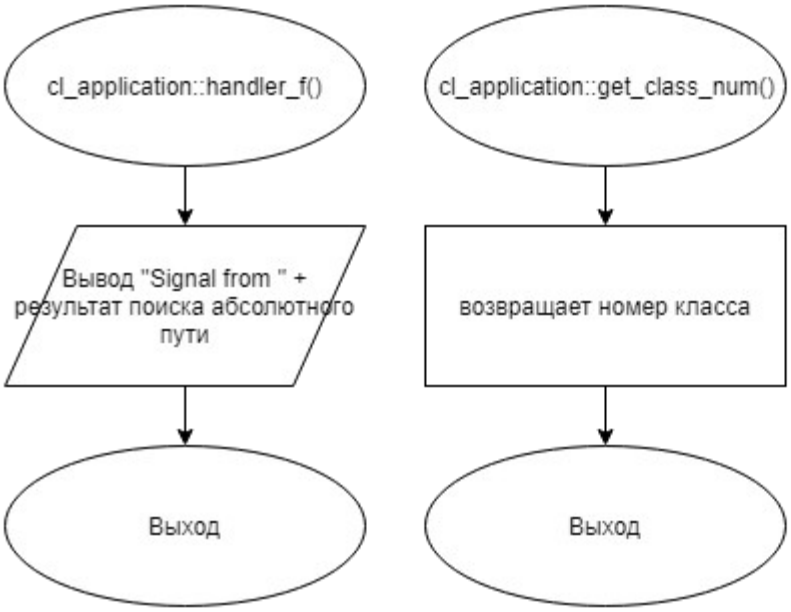

Рисунок 2 – Блок-схема алгоритма

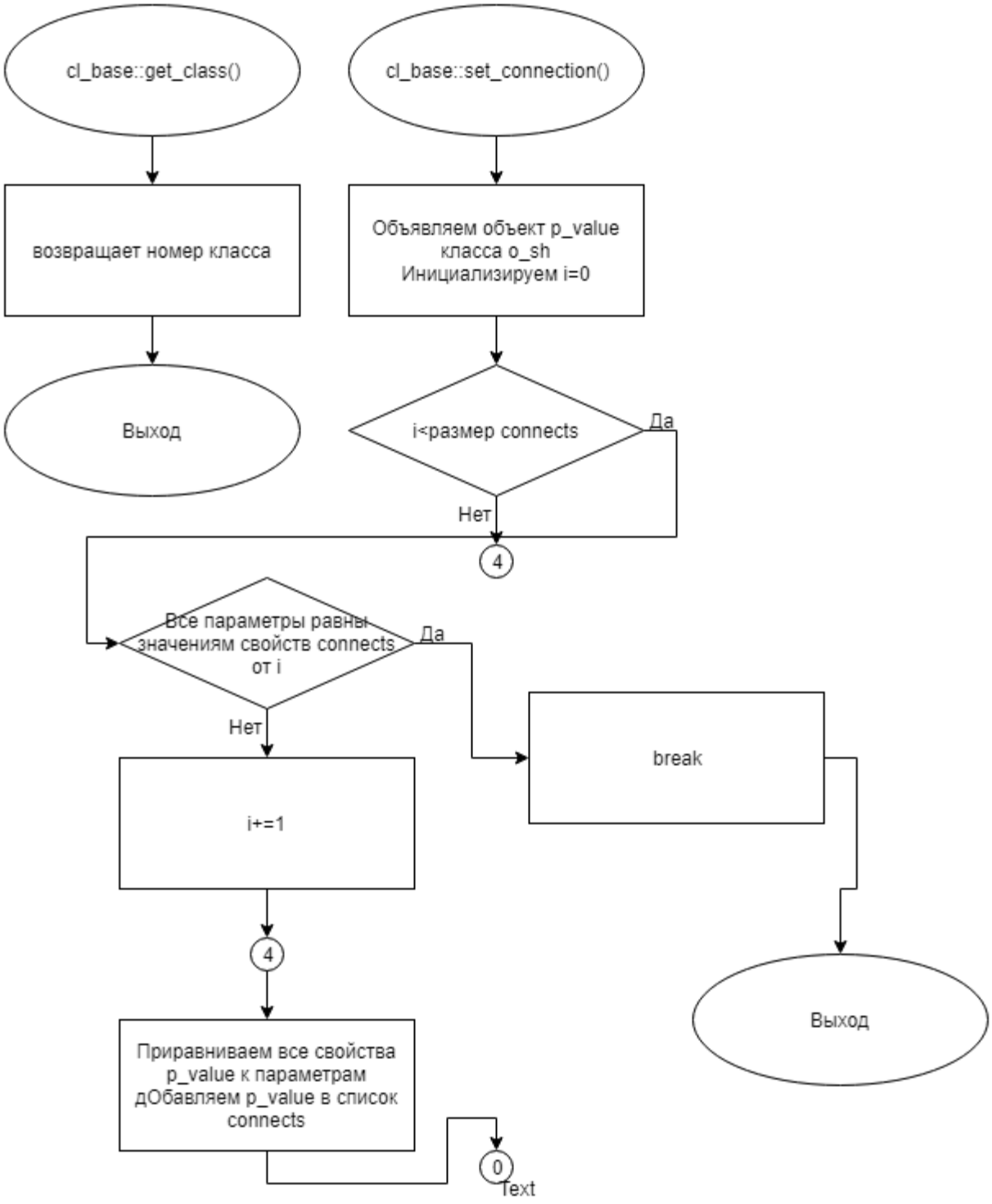

Рисунок 3 – Блок-схема алгоритма

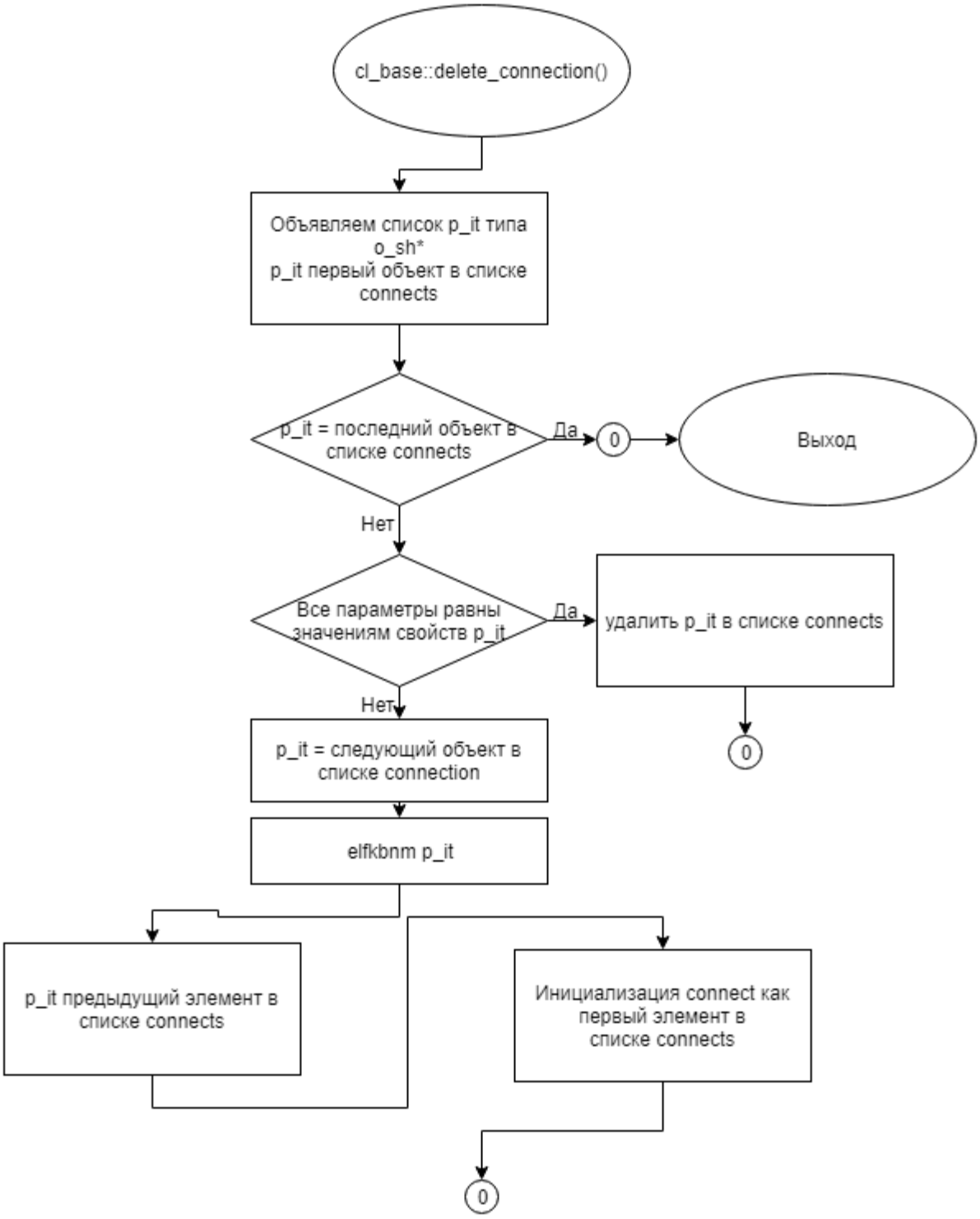

Рисунок 4 – Блок-схема алгоритма

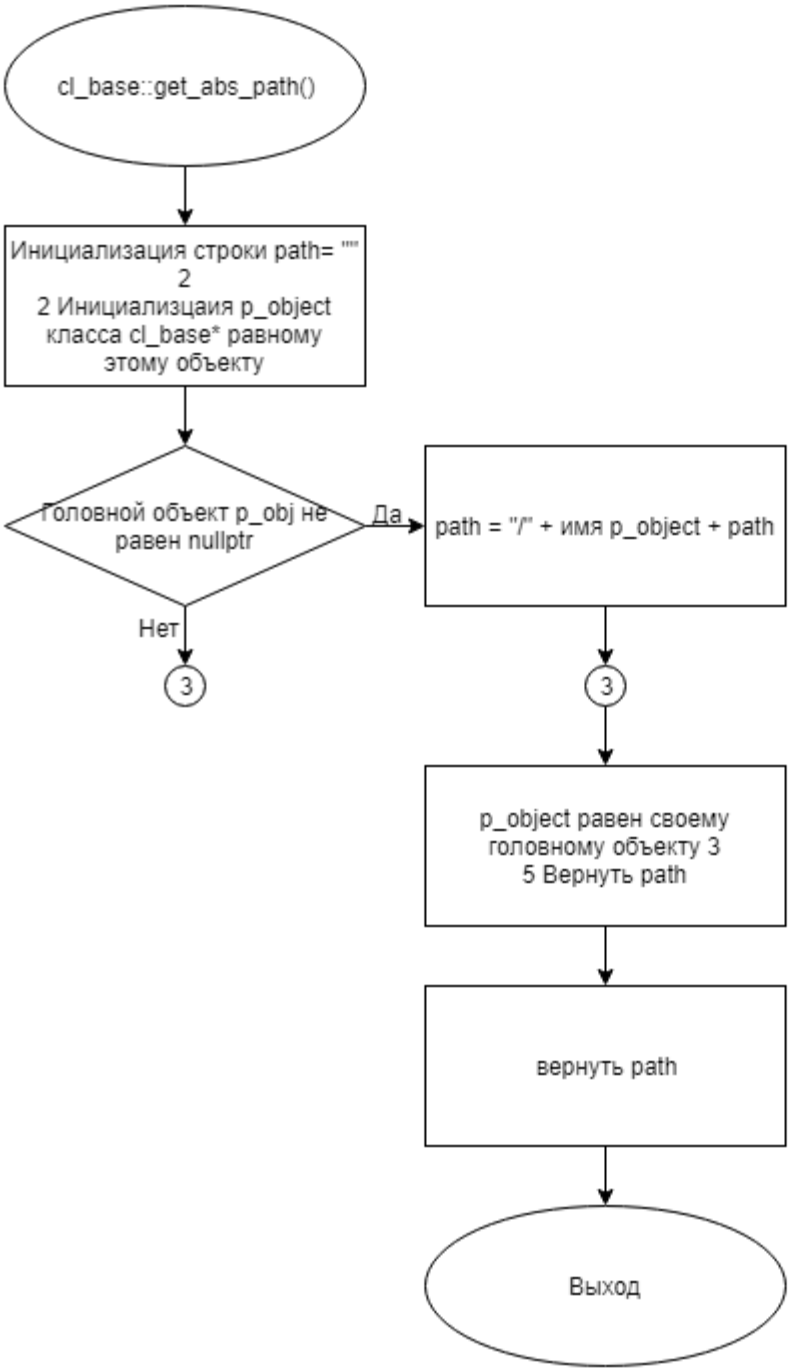

Рисунок 5 – Блок-схема алгоритма

Рисунок 6 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"
cl_2::cl_2(cl_base* p_head_obj, string s_name):cl_base(p_head_obj,s_name){}

int cl_2::get_class_num() {
   return 2;
}
void cl_2::signal_f(string & msg) {
   cout<<"Signal from "<<get_abs_path()<<endl;
   msg+=" (class: 2)";
}
void cl_2::handler_f(string msg) {
   cout<<"Signal to "<<get_abs_path()<<" Text:  " << msg<<endl;
}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include <iostream>
#include "cl_base.h"
class cl_2: public cl_base
{
public:
   cl_2(cl_base* p_head_obj, string s_name);
   void signal_f(string &msg);
   void handler_f(string msg);
   int get_class_num();
};
#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(cl_base* p_head_obj, string s_name):cl_base(p_head_obj,s_name){}

int cl_3::get_class_num() {
   return 3;
}
void cl_3::signal_f(string & msg) {
   cout<<"Signal from "<<get_abs_path()<<endl;
   msg+=" (class: 3)";
}
void cl_3::handler_f(string msg) {
   cout<<"Signal to "<<get_abs_path()<<" Text:  " << msg<<endl;
}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include <iostream>
#include "cl_base.h"
class cl_3: public cl_base
{
public:
   cl_3(cl_base* p_head_obj, string s_name);
   void signal_f(string &msg);
   void handler_f(string msg);
   int get_class_num();
};
#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(cl_base* p_head_obj, string s_name):cl_base(p_head_obj,s_name){}
```

```
int cl_4::get_class_num() {
   return 4;
}
void cl_4::signal_f(string & msg) {
   cout<<"Signal from "<<get_abs_path()<<endl;
   msg+=" (class: 4)";
}
void cl_4::handler_f(string msg) {
   cout<<"Signal to "<<get_abs_path()<<" Text:  " << msg<<endl;
}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include <iostream>
#include "cl_base.h"
class cl_4: public cl_base
{
public:
   cl_4(cl_base* p_head_obj, string s_name);
   void signal_f(string &msg);
   void handler_f(string msg);
   int get_class_num();
};
#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(cl_base* p_head_obj, string s_name):cl_base(p_head_obj,s_name){}

int cl_5::get_class_num() {
   return 5;
}
void cl_5::signal_f(string & msg) {
   cout<<"Signal from "<<get_abs_path()<<endl;
   msg+=" (class: 5)";
}
void cl_5::handler_f(string msg) {
   cout<<"Signal to "<<get_abs_path()<<" Text:  " << msg<<endl;
```

```
}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include <iostream>
#include "cl_base.h"
class cl_5: public cl_base
{
public:
   cl_5(cl_base* p_head_obj, string s_name);
   void signal_f(string &msg);
   void handler_f(string msg);
   int get_class_num();
};
#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"
cl_6::cl_6(cl_base* p_head_obj, string s_name):cl_base(p_head_obj,s_name){}

int cl_6::get_class_num() {
   return 6;
}
void cl_6::signal_f(string & msg) {
   cout<<"Signal from "<<get_abs_path()<<endl;
   msg+=" (class: 6)";
}
void cl_6::handler_f(string msg) {
   cout<<"Signal to "<<get_abs_path()<<" Text:  " << msg<<endl;
}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include <iostream>
#include "cl_base.h"
class cl_6: public cl_base
{
public:
   cl_6(cl_base* p_head_obj, string s_name);
   void signal_f(string &msg);
   void handler_f(string msg);
   int get_class_num();
};
#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <iostream>
using namespace std;
cl_application::cl_application(cl_base* p_head_obj):cl_base(p_head_obj){}
void cl_application::build_tree_objects()
{
   string s_head, s_sub, coord; //имя головного и подчинённого
   cl_base* p_head = this; //указатель на головной объект
   cl_base* p_sub (nullptr); //указатель на подчинённый объект
   cin >> s_head;
   this->set_name(s_head);
   //p_head = this;
   //Новые пер. - номер класса и номер состояния
   int i_class, i_state;
   //int i_class;
   while(true)
   {
      cin >> s_head;
      if(s_head=="endtree")
         break;
      p_head=find_obj_by_coord(s_head);
```

```
         if(p_head!=nullptr)
         {
            cin>> s_sub >> i_class;
            if(!p_head->get_sub_object(s_sub))
            {
               switch(i_class)
               {
                  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<<s_head<<" Dubbing the names of subordinate objects"<<endl;
               exit(0);
            }
         }
         else {
            cout<<"Object tree";
            print_tree();
            cout << endl;
            cout<<"The head object "<<s_head<<" is not found"<<endl;
            exit(0);
         }
      }

   vector<TYPE_SIGNAL>signals={SIGNAL_D(cl_application::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),S
   IGNAL_D(cl_6::signal_f)};
      vector<TYPE_HANDLER>

      handlers={HANDLER_D(cl_application::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)};
      string s_sender,s_receiver,com;
      cl_base *p_sender,*p_receiver;
      while (true) {
         cin>>com;
         if (com=="end_of_connections") {
            break;
         }
```

```
            s_sender=com;
            p_sender=find_obj_by_coord(s_sender);
            if (p_sender!=nullptr) {
               cin>>s_receiver;
               p_receiver=find_obj_by_coord(s_receiver);
               if (p_receiver!=nullptr) {
                  p_sender->set_connection(signals[p_sender->get_class_num()-
      1],p_receiver,handlers[p_receiver->get_class_num()-1]);
               }
               else
               {
                  cout<<"Handler object "<<s_receiver<<" not found"<<endl;
               }
            }
            else
            {
               cout<<"Object "<<s_sender<<" not found"<<endl;
            }
         }
      }

      int cl_application::exec_app()
      {
         cout<<"Object tree";
         print_tree();
         cout << endl;
         string command,coord,text,s_sender,s_receiver;
         cl_base *p_sender=nullptr;
         cl_base* p_receiver=nullptr;
         int i_state;
         vector<TYPE_SIGNAL>
            signals={SIGNAL_D(cl_application::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={HANDLER_D(cl_application::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)};
         this->turn_on_subtree();
         while (true) {
            getline(cin,command);
            if (command=="END") {
               break;
            }
            if (command[0]=='E') {
               s_sender=command.substr(5,command.find(" ",5)-5);
               text=command.substr(command.find(" ",5)+1);
               p_sender=find_obj_by_coord(s_sender);
               if (p_sender!=nullptr) {
                  p_sender->emit_signal(signals[p_sender->get_class_num()-1],text);
               }
               else {
                  cout<<"Object "<<s_sender<<" not found"<<endl;
```

```
            }
         }
         else if (command[7]=='N') {
            s_sender=command.substr(12,command.find(" ",12)-12);
            p_sender=find_obj_by_coord(s_sender);
            if (p_sender!=nullptr) {
               s_receiver=command.substr(command.find(" ",12)+1);
               p_receiver=find_obj_by_coord(s_receiver);
               if (p_receiver!=nullptr) {
                  p_sender->set_connection(signals[p_sender->get_class_num()-
   1],p_receiver,handlers[p_receiver->get_class_num()-1]);
               }
               else {
                  cout<<"Handler object "<<s_receiver<<" not found"<<endl;
               }
            }
            else {
               cout<<"Object "<<s_sender<<" not found"<<endl;
            }
         }
         else if (command[0]=='D') {
            s_sender=command.substr(15,command.find(" ",15)-15);
            p_sender=find_obj_by_coord(s_sender);
            if (p_sender!=nullptr) {
               s_receiver=command.substr(command.find(" ",15)+1);
               p_receiver=find_obj_by_coord(s_receiver);
               if (p_receiver!=nullptr) {
                  p_sender->delete_connection(signals[p_sender->get_class_num()-
   1],p_receiver,handlers[p_receiver->get_class_num()-1]);
               }
               else {
                  cout<<"Handler object "<<s_receiver<<" not found"<<endl;
               }
            }
            else {
               cout<<"Object "<<s_sender<<" not found"<<endl;
            }
         }
         else if (command[7]=='D') {
            s_sender=command.substr(14,command.find(" ",14)-14);
            p_sender=find_obj_by_coord(s_sender);
            i_state=atoi(command.substr(command.find(" ",14)+1).c_str());
            if (p_sender!=nullptr) {
               p_sender->set_state(i_state);
            }
            else {
               cout<<"Object "<<s_sender<<" not found"<<endl;
            }
         }
      }
      return 0;
   }
   int cl_application::get_class_num()
   {
```

```
   return 1;
}
void cl_application::signal_f(string & msg) {
   cout<<"Signal from "<<get_abs_path()<<endl;
   msg+=" (class: 1)";
}
void cl_application::handler_f(string msg) {
   cout<<"Signal to "<<get_abs_path()<<" Text:  "<<msg<<endl;
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H
#include "cl_base.h"
   //добавить остальные 5 классов
   class cl_application: public cl_base
   {
   public:
   cl_application(cl_base* p_head_obj);
   //void build_tree_objects(); В этой курсовой меняем этот метод (меняется
метод создания объектов)
   int exec_app();
   void build_tree_objects();
   void signal_f(string& msg);//наваш
   void handler_f(string msg);//наваш
   int get_class_num();//наваш
   };
#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
   #include <iostream>
   #include <vector>
   #include <string>
   using namespace std;
cl_base::cl_base(cl_base* p_head_obj, string s_name)
{
   //(*this).s_name
   //тем самым обращаемся к полю а не параметру
```

```
   this->s_name = s_name;
   this->p_head_obj = p_head_obj;
   if(this->p_head_obj != nullptr)
      p_head_obj->p_sub_objects.push_back(this);
}
cl_base::~cl_base()
{
   for (int i{}; i<p_sub_objects.size(); i++)
      delete p_sub_objects[i];
}
bool cl_base::set_name(string s_new_name)
{
   /*
   Описание: Новое имя объекта в дереве.
   Параметры: s_new_name - новое имя узла дерева
   */
   if(this->p_head_obj != nullptr)
      for(int i{}; i<p_head_obj->p_sub_objects.size(); i++)

         if(p_head_obj->p_sub_objects[i]->get_name()==s_new_name)
            return false;
   this->s_name = s_new_name;
   return true;
}
string cl_base::get_name()
{
   return this -> s_name;
}
cl_base* cl_base::get_head_obj()
{
   return this -> p_head_obj;
}
cl_base* cl_base::get_sub_object(string s_name)
{
   for(int i{}; i<p_sub_objects.size(); i++)
      if(p_sub_objects[i]->get_name() == s_name)
         return p_sub_objects[i];
   return nullptr;
}
/*void cl_base::print_tree() Обновлённая версия ниже
{
//Описание: Вывод дерева иерархии объектов
//Параметры: -
if(p_sub_objects.size() != 0)
{
cout << endl << get_name();
for(int i{}; i<p_sub_objects.size(); i++)
{
cout << " " << p_sub_objects[i]->get_name();
p_sub_objects[i]->print_tree();
}
}
}*/
//2 курсовая изменения!
//Использует  динамическую  структуру  данных  "очередь".  Он  похож  на
```

```
стек(Last_In_First_Out), но работает по принципу (FIFO)
   //Для использования "очереди" подключаем стандартную библиотеку шаблонов
stl- #include <queue>
cl_base* cl_base::search_current(string s_name) //(рекурсивный метод)обхода
графа в ширину
   {
   //(условно в том порядке, в котором мы добавляем элементы, в том же
проверяем, поэтому так называется)
   /*
   Описание: Поиск объекта по имени в поддереве.
   Параметры: s_name - имя искомого объекта
   */
   cl_base* p_found = nullptr; //Указатель на объект, который мы нашли
   queue<cl_base*> q; //Создаём очередь элеменетов типа cl_base*
   q.push(this);
   while(!q.empty()) //Проверяем элемент на вершине очереди и сравниваем с
тем который мы ищем, пока "очередь" не пуста
   {
   cl_base* p_front = q.front(); //Указатель на первый элемент "очереди".
(функция Front возвращает указатель на первый элемент очереди)
   //q.pop(); //pop удаляет элемент из начала "очереди"
   //Сравниваем имя текущего элемента(p_front) с именем, который мы ищем
   if(p_front->get_name()==s_name)
   if(p_found==nullptr)
   p_found=p_front;
   else
   return nullptr;
   //Теперь перед новой итерацией добавляем в "очередь" дочерние элементы
главного p_front
   for(auto p_sub: p_front->p_sub_objects)
   q.push(p_sub);
   q.pop();
   }
   return p_found;
   }
   cl_base* cl_base::search_tree(string s_name)
   {
      /*
      Описание: Поиск объекта по имени во всём дереве.
      Параметры: s_name - имя искомого объекта
      */
      if(p_head_obj != nullptr)
         return p_head_obj->search_tree(s_name);
      else
         return search_current(s_name);
      /*cl_base* p_root = this;
      while(p_root -> get_head_obj() != nullptr) //Пока вышестоящий объект не
      пустой, мы поднимаемся наверх
      p_root = p_root->get_head_obj();
      return   p_root->search_current(s_name);   //Вызываем   метод   поиска
элемента*/
   }
//Теперь опишем метод состояния готовности объекта
void cl_base::set_state(int status)
{
```

```
   if(p_head_obj ==nullptr || p_head_obj->status != 0)
   {
      this->status = status;

   }
   if(status == 0)
   {
      this->status = status;
      for(int i{}; i < p_sub_objects.size(); i++)
         p_sub_objects[i]->set_state(status);
   }
   /*if(state != 0) //Если так, то проверяем все выше стоящие элементы
   включены ли они
   {
   if(p_head_object !=nullptr && p_head_object->status != 0)
   {
   status = state;
   }
   }
   else //иначе обходим и выключаем все подчинённые элементы (рекурсивно)
   {
   status = state;
   for (int i = 0; i<p_sub_objects.size(); i++)
   {
   p_sub_objects[i]->set_state(state);
   }
   }*/
}
void cl_base::print_tree(int i_space)
{
   //Описание: Вывод дерева иерархии объектов
   //Параметры: int i_space
   cout << endl << string(i_space, ' ') << this->get_name();
   //for(auto p_sub : p_sub_objects)
   //p_sub->print_tree(i_space + 4);
   for(int i{}; i<p_sub_objects.size(); i++)
      p_sub_objects[i]->print_tree(i_space + 4);
}

void cl_base::print_tree_state(int i_space)
{
   //Описание: Вывод дерева иерархии объектов
   //Параметры: int i_space
   cout << endl << string(i_space, ' ');
   //cout <<<< this->get_name();
   if(this->status != 0)
      cout << this-> get_name() << " is ready";
   else
      cout << this-> get_name() << " is not ready";
   //for(auto p_sub : p_sub_objects)
   //p_sub->print_tree_state(i_space + 4);
   for(int i{}; i<p_sub_objects.size(); ++i)
      p_sub_objects[i]->print_tree_state(i_space + 4);
}
/* 3 курсовая */
```

```
cl_base* cl_base :: find_obj_by_coord(string s_coord)
{
   if (s_coord.empty())
   {
      return nullptr;
   }
   cl_base* temp = this;
   if (s_coord[0] == '.')
   {
      if (s_coord == ".")
      {
         return temp;
      }
      s_coord.erase(0, 1);
      return temp -> search_current(s_coord);
   }
   else if (s_coord[0] == '/')
   {
      while (temp -> get_head_obj())
      {
         temp = temp -> get_head_obj();
      }
      if (s_coord == "/")
      {
         return temp;
      }
      if (s_coord[1] == '/')
      {
         s_coord.erase(0, 2);
         return temp -> search_tree(s_coord);
      }
      s_coord.erase(0, 1);
   }
   string s_name;
   for (int i = 0; i < s_coord.length(); i++)
   {
      if (s_coord[i] == '/')
      {
         temp = temp -> get_sub_object(s_name);
         if (temp == nullptr)
         {
            return temp;
         }
         s_name = "";
      }
      else
      {
         s_name += s_coord[i];
      }
   }
   return temp -> get_sub_object(s_name);
}

void cl_base::delete_sub_obj(string s_name)
{
```

```
   for (int i{}; i<p_sub_objects.size(); i++)
   {
      if(p_sub_objects[i]->get_name()==s_name)
      {
         p_sub_objects.erase(p_sub_objects.begin()+i);
         i--;
      }
   }
}

bool cl_base::set_head_obj(cl_base* p_new_head)
{
   cl_base* p_pobj = p_new_head;
   if (p_new_head == nullptr)
   {
      return false;
   }
   if (!this -> p_head_obj)
   {
      return false;
   }
   if (p_new_head -> get_sub_object(this -> get_name()))
   {
      return false;
   }
   while (p_pobj != nullptr)
   {
      if (p_pobj == this)
      {
         return false;
      }
      else
      {
         p_pobj = p_pobj -> get_head_obj();
      }
   }
   this->get_head_obj()->delete_sub_obj(this->get_name());
   this->p_head_obj=p_new_head;
   this->p_head_obj->p_sub_objects.push_back(this);
   return true;
}
//чор
void          cl_base::set_connection(TYPE_SIGNAL          p_signal,cl_base*
p_target,TYPE_HANDLER p_handler)
//для установления соединения
//Вызов метода сигнала с передачей строковой переменной по ссылке.
{
   o_sh* p_value;
   for (int i = 0;i<connects.size();i++) {
      if                        (connects[i]->p_signal==p_signal&&connects[i]-
>p_target==p_target&&connects[i]->p_handler==p_handler) {
         return;
      }
   }
   p_value=new o_sh();
```

```
   p_value->p_signal=p_signal;
   p_value->p_target=p_target;
   p_value->p_handler=p_handler;
   connects.push_back(p_value);
}
void                                    cl_base::delete_connection(TYPE_SIGNAL
p_signal,cl_base*p_target,TYPE_HANDLER p_handler)
//для удаления соединения
//Удаления (разрыва) связи между сигналом текущего объекта и обработчиком
целевого объекта.
{
   vector<o_sh*>::iterator p_it;//объявляем список
   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) {
         //если p_it последний объект в списке connects
         delete *p_it;//удаляет объект
         p_it=connects.erase(p_it);
         p_it--;
      }
   }
}
int cl_base::get_state() {
   return status;
}
void cl_base::emit_signal(TYPE_SIGNAL p_signal,string s_message) {
//Выдачи сигнала от текущего объекта с передачей строковой переменной.
   if (this->get_state()!=0) {
      (this->*p_signal)(s_message);
      for (auto connection:connects) {
         if (connection->p_signal==p_signal) {
            cl_base* p_target=connection->p_target;
            TYPE_HANDLER p_handler = connection->p_handler;
            if (p_target->get_state()!=0) {
               (p_target->*p_handler)(s_message);

            }
         }
      }
   }
}
string cl_base::get_abs_path () {
   //абсолютный путь
   string path="";
   cl_base* p_object = this;
   while  (p_object->get_head_obj()!=nullptr)  {//головной  объект  не  равен
nullptr
      path="/"+p_object->get_name()+path;
      p_object=p_object->get_head_obj();//равен своему головному объекту
   }
   if (path=="") {
      return "/";
   }
   return path;
}
```

```
void cl_base::turn_on_subtree() {
   this->set_state(1);
   for (auto p_sub:this->p_sub_objects) {
      p_sub->turn_on_subtree();
   }
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <vector>
#include <string>
#include <queue>
using namespace std;
class cl_base;
#define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f)
typedef void(cl_base::*TYPE_SIGNAL) (string & msg);
typedef void(cl_base::*TYPE_HANDLER) (string msg);
struct o_sh
{
   TYPE_SIGNAL p_signal;
   cl_base* p_target;
   TYPE_HANDLER p_handler;
};

class cl_base
{
   string s_name;
   cl_base* p_head_obj = nullptr;
   vector<cl_base*> p_sub_objects;
   vector<o_sh*> connects; //Вектор соединений
   int status = 0;
public:
   cl_base(cl_base* p_head_obj, string s_name = "Base object");
   bool set_name(string s_new_name);
   string get_name();
   cl_base* get_head_obj();

   cl_base* get_sub_object(string s_name);
   ~cl_base();
   cl_base* search_current(string); //ищет по текущему
   cl_base* search_tree(string); //ищет по всему дереву
   void set_state(int);

   int get_state();
```

```
   void print_tree(int i_space= 0);
   void print_tree_state(int i_space= 0);
   cl_base* find_obj_by_coord(string);

   void delete_sub_obj(string);

   bool set_head_obj(cl_base*);
   virtual int get_class_num()=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 s_message);
   void delete_links(cl_base* p_target);
   void turn_on_subtree();
   string get_abs_path();
   int get_class();
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "cl_application.h"
int main()
{
   cl_application ob_cl_application(nullptr); //создание корневого объекта
   ob_cl_application.build_tree_objects();     //конструирование     системы,
построения дерева
   return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 10.

Таблица 10 – Результат тестирования программы

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

ЗАКЛЮЧЕНИЕ

В ходе данного курса мы углубленно изучили основы объектно-ориентированного программирования, освоили ключевые принципы и понятия этой парадигмы. Мы приступили к созданию собственных программных решений, применяя полученные знания на практике.

Изученные концепции и методы в объектно-ориентированном программировании открывают перед нами множество возможностей для разработки качественного и эффективного программного обеспечения. Эта парадигма поможет нам в создании более надежных и масштабируемых проектов в будущем.

Важным достижением курса стало получение практического опыта разработки программ на популярном языке программирования c++. Этот опыт позволит нам более уверенно и профессионально применять знания в области объектно-ориентированного программирования на практике.

Мы готовы к дальнейшему профессиональному росту и к созданию инновационных и качественных программных продуктов, используя принципы объектно-ориентированной парадигмы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

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

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

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

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

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