Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектно-ориентированное программирование - это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Каждый объект обладает состоянием и поведением, а также идентичностью; структура и поведение схожих объектов определяет общий для них класс;

Используя большое количество ничем не связанных классов управлять имим становится очень сложно. Для решения данной проблемы существует наследование классов. Наследование позволяет упорядочить все схожие свойства и методы разных классов в один базовый. Совокупность всех объектов классов в одной системе и их взаимодействие составляют концепцию ООП.

ООП до сих пор является самым удобным способом написания программы в наше время, хоть сама концепция и была придумана еще в 20 веке.

1 ПОСТАНОВКА ЗАДАЧИ

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задается в следующем виде:

/ - корневой объект;

//«имя объекта» - поиск объекта по уникальному имени от корневого (для однозначности уникальность требуется в рамках дерева);

. - текущий объект;

«имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;

/«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от корневого объекта.

Примеры координат:

/

//ob_3

.

ob_2/ob_3

ob_2

/ob_1/ob_2/ob_3

Если координата пустая строка или объект не найден, то вернуть нулевой указатель.

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы.

Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта соблюдены.

Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов).

Система отрабатывает следующие команды:

SET «координата» – устанавливает текущий объект;

FIND «координата» – находит объект относительно текущего;

END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим.

При вводе данных в названии команд ошибок нет. Условия уникальности имен объектов для однозначной отработки соответствующих команд соблюдены.

1.1 Описание входных данных

Состав и иерархия объектов строится посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы.

Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

SET «координата» - установить текущий объект;

FIND «координата» - найти объект относительно текущего;

END – завершить функционирование системы (выполнение программы).

Команды SET и FIND вводятся произвольное число раз. Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов.

root
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4 object_7 3
endtree
FIND object_2/object_4
SET /object_2
FIND //object_5
FIND /object_15
FIND .

FIND object_4/object_7

END

1.2 Описание выходных данных

Первая строка:

Object tree

Со второй строки вывести иерархию построенного дерева как в курсовой работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение:

The head object «координата головного объекта» is not found

и прекратить работу программы.

Если дерево построено, то далее построчно:

для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

Object is not found: «имя текущего объекта» «искомая координата объекта»

для команд FIND вывести:

«искомая координата объекта» Object name: «наименование объекта»

Если объект не найден, то:

«искомая координата объекта» Object is not found

Пример вывода иерархии дерева объектов.

```
Object tree
root
    object_1
        object_7
    object_2
        object_4
            object_7
        object_5
        object_3
    object_3
object_2/object_4     Object name: object_4
Object is set: object_2
//object_5     Object name: object_5
/object_15     Object is not found
.     Object name: object_2
object_4/object_7     Object name: object_7
```

2 МЕТОД РЕШЕНИЯ

Объекты стандартного потока ввода и вывода данных - cin/cout

Оператор сножественного выбора - switch

Условный оператор - if ... else

Оператор цикла со счетчиком - for

Оператор цикла с предусловием - while

Таблица №1 - Иерархия наследования классов

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы - наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base			Базовый класс, на основании которого строятся классы, из которых состоит дерево иерархии		
		node6	public		7	

Продолжение таблицы 1

		node5	public		6	
		node4	public		5	
		node3	public		4	
		node2	public		3	
		application	public		2	
2	application			Класс корневого объекта		
3	node2			Класс подчиненных объектов		
4	node3			Класс подчиненных объектов		
5	node4			Класс подчиненных объектов		
6	node5			Класс подчиненных объектов		
7	node6			Класс подчиненных объектов		

Класс Base:

1. Методы:

- Метод find_ptr

1. Функционал - получение указателя на любой объект в составе дерева иерархииобъектов согласно пути (координаты)

- Метод find_name

1. Функционал - поиск объекта по имени локального среди подчиненных конкретного объекта

Класс application:

Свойства/поля:

Поле, которое хранит успешное построение дерева:

- Имя - tree
- Тип - булевое значение
- Модификатор доступа - private
- Возвращаемое значение - поле, которое хранит сообщение об ошибке

5. Имя - error
6. Тип - строковое значение
7. Модификатор доступа - private

Методы:

- Метод build_tree

1. Функционал - построение дерева, где ввод организован как в версии №2 в курсовой работы. Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

- Метод start

1. Функционал - запуск приложения. Если собрать дерево невозможно, прекращает сборку и выводит иерархию уже построенного фрагмента дерева и завершает работу программы.

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 метода find_ptr класса base

Функционал: получение указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты).

Параметры: переменная строкового типа way.

Возвращаемое значение: указатель на класс base.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода find_ptr класса base

№	Предикат	Действия	№ перехода
1	way - пустая переменная	Возврат nullptr	∅
			2
2	way начинается с '.' и имеет только 1 символ	Возврат указателя на текущий объект	∅
			3
3		Объявление указателя ptr на класс base	4
4		Присвоение указателю ptr указатель на текущий объект	5
5	way начинается с '/'		6
			10
6	У данного объекта есть головной объект	Присвоение ptr указатель на головной объект	6
			7
7	way состоит из одного		∅

Продолжение таблицы 2

№	Предикат	Действия	№ перехода
	символа		
			8
8	Второй символ = '/'	Удаление первых двух символов переменной way	9
		Удаление первого символа переменной way	10
9		Возврат результата работы метода find	∅
10		Объявление перменной s	11
11		Объявление переменной f_name	12
12	s = -1	Присвоение f_name значение way	13
		Присвоение f_name значение way с s'го символа	13
13	Обход вектора подчиненных элементов объекта ptr не закончен		14
		Возврат nullptr	∅
14	Имя объекта sub совпадает с f_name		15
			13
15	s = -1	Возврат sub	∅
		Получение из переменной way название объекта	16
16		Возврат результата работы метода find_name	∅

3.2 Алгоритм метода build_tree класса application

Функционал: построение дерева, где ввод организован, как в версии №2 курсовой работы. Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный к нему..

Параметры: отсутствуют

Возвращаемое значение: отсутствуют.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода build_tree класса application

№	Предикат	Действия	№ перехода
1		Обявление сроковых переменных main, head, sub	2
2		Объявление указателя ptr на класс base	3
3		Обявление целочисленной переменной number	4
4		Ввод значения переменной main	5
5		Вызов метода set_name	6
6	true	Ввод значения переменной head	7
			∅
7	head = "endtree"	break	∅
			8
8		Ввод значений переменных sub и number	9
9		Присвоение ptr результата работы метода find с параметром head	10
10	ptr != нулевому указателю		11
		Привоение полю tree значение false	12
11	number == 2	Создание объекта типа node2 с параметрами ptr и sub	6
	number == 3	Создание объекта типа node3 с параметрами ptr и sub	6

Продолжение таблицы 3

	number == 4	*Создание объекта типа node4 с параметрами ptr и sub*	*6*
	number == 5	*Создание объекта типа node4 с параметрами ptr и sub*	*6*
	number == 6	*Создание объекта типа node4 с параметрами ptr и sub*	*6*
12		*Присвоение полю error значение "The head object " значение переменной head " is not found"*	∅

3.3 Алгоритм метода find_name класса base

Функционал: поиск объекта по имени локального среди подчиненных конкректного объекта.

Параметры: переменная строкового типа name_.

Возвращаемое значение: указаьель на класс base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода find_name класса base

№	Предикат	Действия	№ перехода
1	name_ = полю name	Возврат указателя на текущий объект	∅
		Объявление целочисленной переменной счетчика i	2
2	i < размера вектора подчиненных элементов		3
		Возврат нулевого указателя	∅
3	Имя i'го элемента = name_	Возврат i'го указателя	∅
		Постфиксный инкремент переменной i	2

3.4 Алгоритм метода start класса application

Функционал: запуск приложения. Если собрать дерево невозможно, прекращает сборку и выводит иерархию уже построенного фрагмента дерева и завершает работу программы..

Параметры: .

Возвращаемое значение: целочисленный флаг успешности выполнения работы.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода start класса application

№	Предикат	Действия	№ перехода
1		Вызов метода output	2
2	ПОле tree имеет значение истина	Обявление указателя ptr на класс base	3
			14
3		Присвоение указателю ptr указатель на текущий объект	4
4		Обявление строковой переменной cmd	5
5	Идет ввод cmd и cmd != "END"	Обявление строковой переменной way	6
			14
6		Ввод значения переменной way	7
7		Вывод перенос строки	8
8	cmd == "SET"	Объявление указателя newptr на класс base и присвоение результата работы методв find_ptr	9
			11
9	newptr != nullptr	Присвоение ptr значение newptr	10
		Вывод "Object is not found: " имя объекта путь объекта	5

Продолжение таблицы 5

10		Ввод "Object is set: " имя объекта	5
11	cmd == "FIND"	Объявление указателя f_ptr на класс base и присвоение результата работы метода find_ptr	12
			5
12		Вывод значенияя переменной way	13
13	f_ptr != nullptr	Вывод отступ "Object name: " имя объекта	5
		Вывод отступ "Object is not found"	5
14		return 1	∅

3.5 Алгоритм функции main

Функционал: главная функция программы.

Параметры: .

Возвращаемое значение: целочисленный флаг успешности выполнения работы.

Алгоритм функции представлен в таблице 6.

Таблица 6 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта a класса application с помощью конструктора с параметром nullptr	2
2		Вызов метода build_tree объектом a	3
3		Вызов метода start объектом a	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-4.

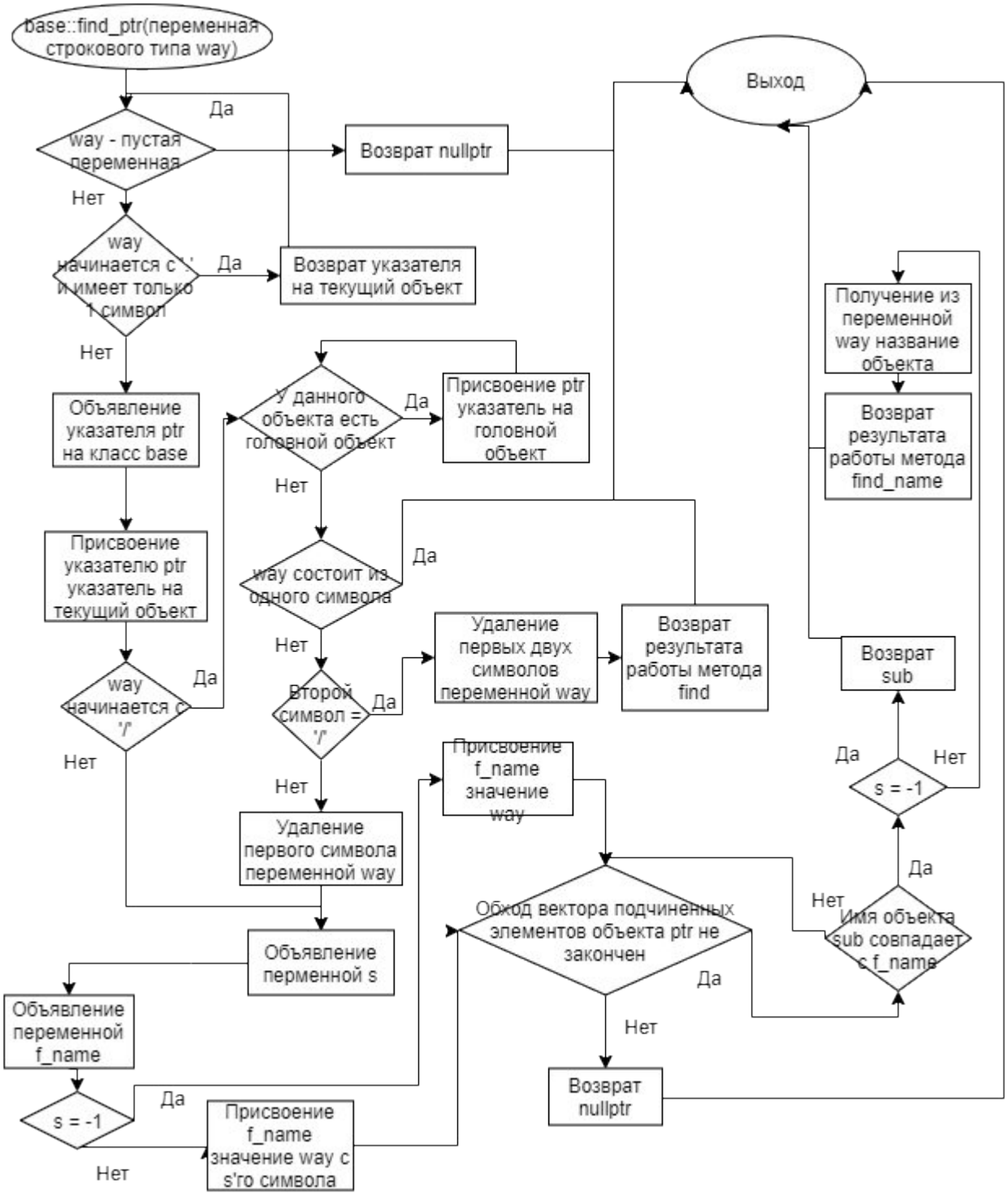

Рисунок 1 – Блок-схема алгоритма

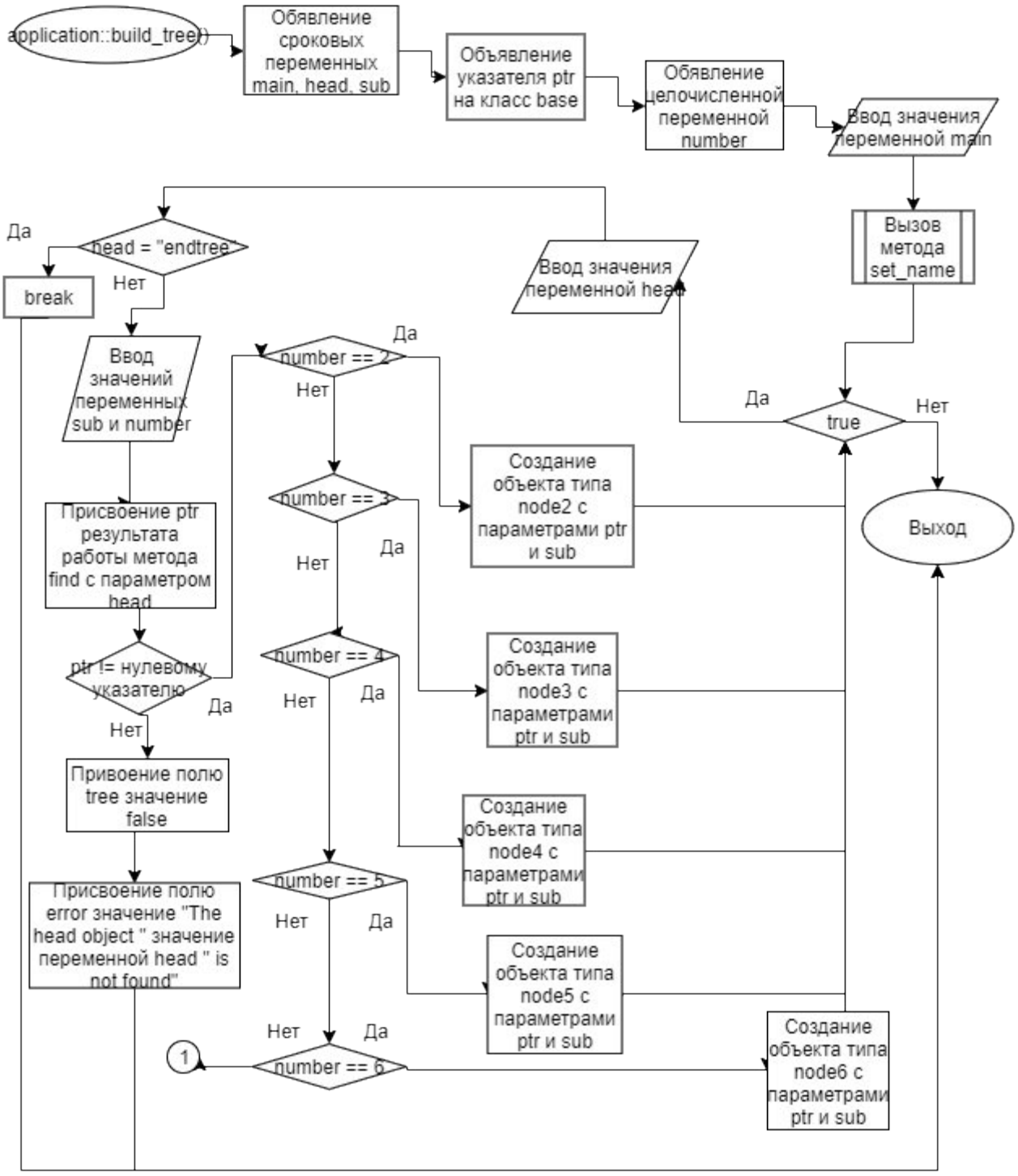

Рисунок 2 – Блок-схема алгоритма

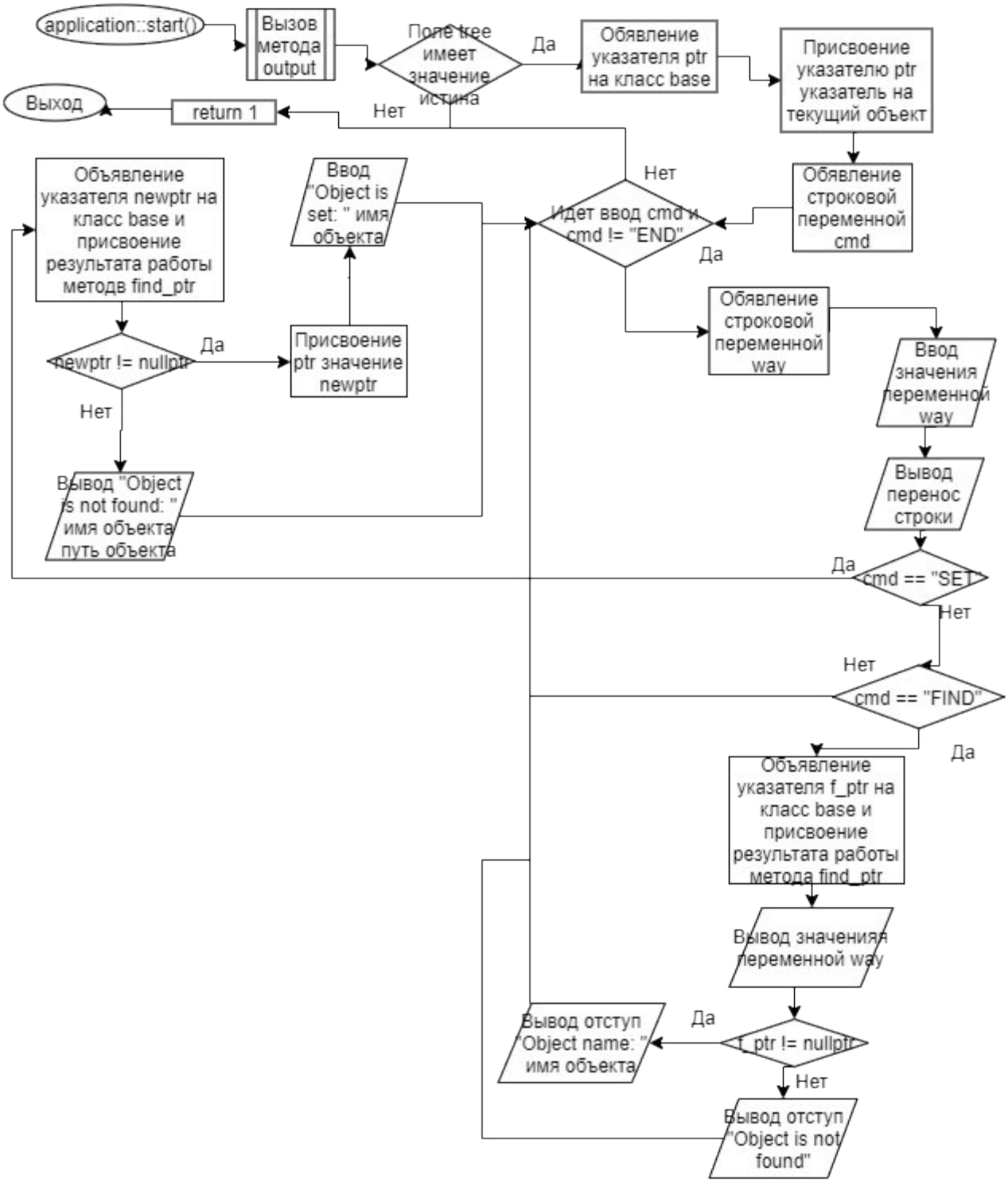

Рисунок 3 – Блок-схема алгоритма

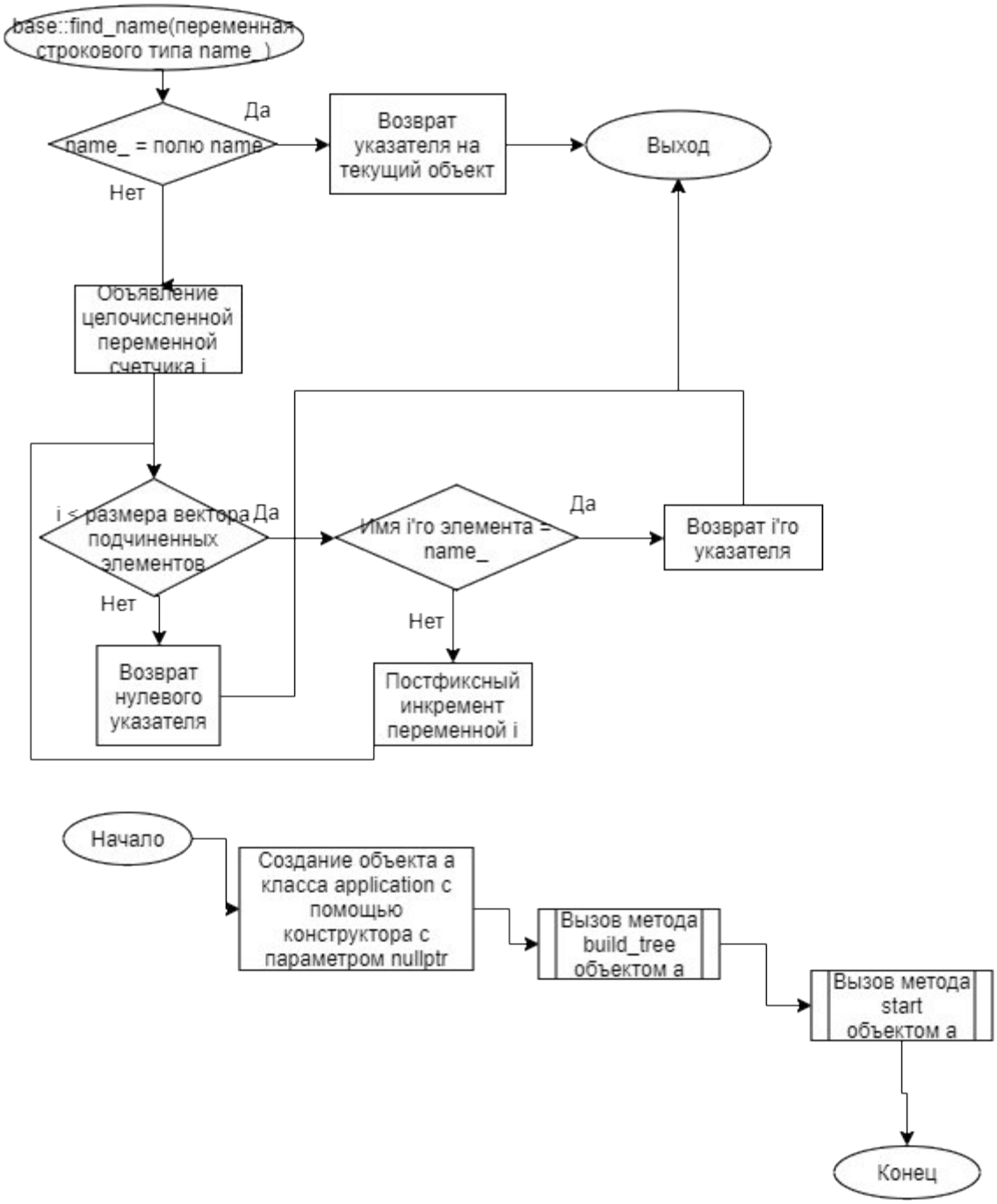

Рисунок 4 – Блок-схема алгоритма

ЗАКЛЮЧЕНИЕ

В рамках выполнения курсовой работы я разработал программу реализации дерева иерархии объектов и взаимодействия с элементами дерева. Я научился:
Разрабатывать базовый класс для объектов;
Настраивать взаимосвязь объектов в дереве иерархии;
Добавлять и удалять объекты в дереве иерархии;
Выстраивать дерево иерархии объектов;

Благодаря ООП я научился выстраивать правильную систему наследования классов. Ознакомился с концепцией построения дерева иерархии. Также выполнение курсовой работы помогло мне обрести новые навыки в программировании.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. 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).

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

Приложение 1. Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

1.1 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"

application::application(Base* ptr):Base(ptr){}
int application::start(){
      output();
      if(tree){
            Base* ptr = this;
            string cmd;
            while(cin >> cmd && cmd !=
                  "END"){string way;
                  cin >> way;
                  cout << endl;
                  if(cmd == "SET"){
                        Base* newptr = ptr -> find_ptr(way);
                        if(newptr != nullptr){
                              ptr = newptr;
                              cout << "Object is set: " << ptr -> get_name();}
                        else{
                              cout << "Object is not found: " << get_name() << " "
                              << way;}}
                  else{
                        if(cmd == "FIND"){
                              Base* f_ptr = ptr -> find_ptr(way);
                              cout << way;
                              if(f_ptr != nullptr){
                                    cout << "       Object name: " << f_ptr ->
                                    get_name();}
                              else{cout << "     Object is not found";}}}}}
      else{cout << endl << error;}
      return 1;}
void
      application::build_tree(){s
      tring main, head, sub;
      Base* ptr;
      int number = 0;
      cin >> main;
      this -> set_name(main);
      while(true){
            cin >> head;
            if(head == "endtree") break;
            cin >> sub >> number;
```

Продолжение листинга 1

```
            if(ptr != nullptr){
                  switch(number){
                        case 1:
                              new node(ptr, sub);
                              break;
                        case 2:
                              new node2(ptr, sub);
                              break;
                        case 3:
                              new node3(ptr, sub);
                              break;
                        case 4:
                              new node4(ptr, sub);
                              break;
                        case 5:
                              new node5(ptr, sub);
                              break;
                        case 6:
                              new node6(ptr, sub);
                              break;
                        default:
                              break;}}
            else{
                  this -> tree = false;
                  this -> error = "The head object " + head + " is not found";
                  return;}}}
```

1.2 Файл application.h

Листинг 2 – application.h

```
#ifndef application_h
#define application_h
#include "node.h"
#include "Base.h"
class application: public
      Base{private:
      string error;
      bool tree = true;
      public:
      application(Base* ptr);
      int start();
      void build_tree();};
#endif
```

1.3 Файл Base.cpp

Листинг 3 – Base.cpp

```
#include "Base.h"
Base::Base(Base* root, string
      obj_name){name =obj_name;
      this -> root = root;
      if(root != nullptr){root -> arr_slave.push_back(this);}}
int a(0), b(0);
void Base::output(){
      if(root == nullptr){
            cout << "Object tree" << endl;
            cout << name;}
      if(arr_slave.size() > 0){
            for(int i = 0; i < arr_slave.size();
                  i++){cout << endl << "";
                  if(a != 0){
                        for(int j = 0; j < b; j++) cout << "    ";}
                  a++;
                  b++;
                  cout << arr_slave[i] -> get_name();
                  arr_slave[i] -> output();
                  a--;
                  b--;}}}
Base* Base::get_ptr(){return root;}
void Base::set_name(string obj_name){name = obj_name;}
void Base::def(Base* ptr){
      if(root != nullptr && ptr != nullptr){
            for(int i = 0; i < (root -> arr_slave.size());
                  i++){if(root -> arr_slave[i] -> name == name){
                        root -> arr_slave.erase(root -> arr_slave.begin()+i);
                        break;}}
            root = ptr;
            root -> arr_slave.push_back(this);}}
string Base::get_name(){return name;}
Base* Base::find(string
      name_f){Base* obj;
      if(name_f == name) return this;
      for(int i = 0; i < arr_slave.size();
            i++){obj = arr_slave[i] ->
            find(name_f); if(obj != nullptr)
            return obj;}
      return nullptr;}
void Base::output_ready(string tab){
      ready != 0 ? cout << get_name() <<" is ready":cout << get_name() << " is not
      ready";
      if(arr_slave.size() !=
            0){tab += " ";
            for(int i = 0; i < arr_slave.size();
                  i++){Base* ptr = arr_slave[i];
                  cout << endl << tab;
                  ptr -> output_ready(tab);}}}
void Base::set_ready_tree(){
      string name;
      int readyy;
```

Продожение листинга 3

```
      while(cin >> name >>
            readyy){ if(find(name) !=
            nullptr){
                  find(name) -> set_ready(readyy);}}}
Base* Base::find_ptr(string way){
      if(way.empty()) return nullptr;
      if(way[0] == '.' && way.size() == 1) return this;
      Base* ptr = this;
      if(way[0] == '/'){
            while(ptr -> get_ptr()){ptr = ptr -> get_ptr();}
            if(way.size() == 1) return ptr;
            if(way[1] == '/'){
                  way.erase(0,2);
                  return ptr -> find(way);}
            way.erase(0,1);}
      int s = way.find('/');
      string f_name;
      if(s == -1){f_name = way;}
      else{f_name = way.substr(0, s);}
      for(auto sub : ptr -> arr_slave){
            if(sub -> get_name() ==
                  f_name){if(s == -
                  1){return sub;} else{
                        way.erase(0, s+1);
                        return sub -> find_name(way);}}}
      return nullptr;}
Base* Base::find_name(string
      name_){ if(name_ == name){return
            for(int      =   0;   i   <   arr_slave.size();   i++){arr_slave[i]   ->

iset_ready(0);}}
            Base* ptr = this -> root;
            while(ptr != nullptr){
                  if(ptr -> ready ==
                        0){ready = 0;
                        return;}
                  ptr = ptr -> root;}
```

1.4 Файл Base.h

Листинг 4 – Base.h

```
#ifndef Base_h
#define Base_h
#include <string>
#include <vector>
#include <iostream>
```

Продолжение листинга 4

```
using namespace std;
class Base{
      private:
      int ready;
      string name;
      Base* root;
      vector <Base*> arr_slave;
      public:
      Base(Base* root, string obj_name = "name");
      void set_name(string obj_name);
      string get_name();
      void output();
      void def(Base* ptr);
      Base* get_ptr();
      Base* find(string name_f);
      void set_ready(int readyy);
      void set_ready_tree();
      void output_ready(string tab);
      Base* find_ptr(string way);
      Base* find_name(string name);};
#endif
```

1.5 Файл main.cpp

Листинг 5 – main.cpp

```
#include "main.h"
int main(){
      application a(nullptr);
      a.build_tree();
      return a.start();}
```

1.6 Файл main.h

Листинг 6 – main.h

```
#ifndef main_h
#define main_h
#include <iostream>
#include "node.h"
#include "application.h"
using namespace std;
#endif
```

1.7 Файл node.cpp

Листинг 7 – node.cpp

```
#include "node.h"
node::node(Base* ptr, string name):Base(ptr, name){}
node2::node2(Base* ptr, string name):Base(ptr, name){}
node3::node3(Base* ptr, string name):Base(ptr, name){}
node4::node4(Base* ptr, string name):Base(ptr, name){}
node5::node5(Base* ptr, string name):Base(ptr, name){}
node6::node6(Base* ptr, string name):Base(ptr, name){}
```

1.8 Файл node.h

Листинг 8 – node.h

```
#ifndef node_h
#define node_h
#include "Base.h"
class node: public
      Base{public:
      node(Base* ptr, string name = "empty");};
class node2: public Base{
      public:
      node2(Base* ptr, string name = "empty");};
class node3: public Base{
      public:
      node3(Base* ptr, string name = "empty");};
class node4: public Base{
      public:
      node4(Base* ptr, string name = "empty");};
class node5: public Base{
      public:
      node5(Base* ptr, string name = "empty");};
class node6: public Base{
      public:
      node6(Base* ptr, string name = "empty");};
#endif
```

Приложение 2. Тестирование

Результат тестирования программы представлен в таблице 7.

Таблица 7 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_4 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND object_2/object_4 SET /object_2 FIND //object_5 FIND /object_15 FIND . FIND object_4/object_7 END	Object tree root object_1 object_7 object_2 object_4 object_7 object_4 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object is not found /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7	Object tree root object_1 object_7 object_2 object_4 object_7 object_4 object_3 object_3 object_2/object_4 Object name: object_4 Object is set: object_2 //object_5 Object is not found /object_15 Object is not found . Object name: object_2 object_4/object_7 Object name: object_7