Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектно-ориентированное программирование, ООП – это одна из парадигм разработки, подразумевающая организацию программного кода, ориентируясь на данные и объекты, а не на функции и логические структуры.

Обычно объекты в подобном коде представляют собой блоки с данными, которые имеют определенный набор характеристик и возможностей. Объект может олицетворять что угодно – от человека с именем, фамилией, номером телефона, паролем и другой информацией до мелкой утилиты с минимумом характеристик из реального мира, но увеличенным набором функций. Объекты могут взаимодействовать друг с другом, пользователем и любыми другими компонентами программы.

Идеология объектно-ориентированного программирования (ООП) разрабатывалась, чтобы связать поведение определенного объекта с его классом. Людям проще воспринимать окружающий мир как объекты, которые поддаются определенной классификации (например, разделение на живую и неживую природу).

ООП в разработке использовался другой подход — процедурный. Программа представляется в нем как набор процедур и функций — подпрограмм, которые выполняют определенный блок кода с нужными входящими данными. Процедурное программирование хорошо подходит для легких программ без сложной структуры. Но если блоки кода большие, а функций сотни, придется редактировать каждую из них, продумывать новую логику. В отличие от процедурного, объектно-ориентированное программирование позволяет вносить изменения один раз — в объект. Именно он — ключевой элемент программы. Все операции представляются как взаимодействие между объектами. При этом код более читаемый и понятный, программа проще масштабируется.

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

В данной задаче используется:

Объеты стандартного потока входа/выхода cin/cout

Условный оператор if

Оператор цикла с предусловием while

Класс cl_base:

- Свойства/поля:
- Метод:
  - метод loccation_objects_way

1. функционал - получение указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты)

Класс cl_application:

3.0Свойства/поля:

Поле, отвечающий за готовность построение дерева иерархии

наименование - success;

тип - string;

модификатор доступа - закрытый;

3.1Методы:

Метод bild_tree_objects

функционал - построение дерева иерархии объектов

Метод exec_app

функционал - запуск системы;

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.2 Алгоритм функции main

Функционал: Точка входа в программу, реализация вызова основных методов и инициализация объектов.

Параметры: нет.

Возвращаемое значение: Целое число - индикатор успешного завершения программы.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создаем объект ob_cl_application класса cl_application посредством параметризиравного конструктор с параметром nullptr	2
2		Вызов метода bild_tree_objects_new2 объекта ob_cl_application	3
3		Возвращается результат вызова метода exec_app объекта ob_cl_application	∅

3.3 Алгоритм метода exec_app класса cl_application

Функционал: запуск системы.

Параметры: нет.

Возвращаемое значение: int, нуль или код ошибки.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Инициализация переменоой out типа string и присвоение ей значение ""	2
2		Вывод "Object tree"	3
3		Вызов метода new_print()	4
4		Объявление переменных command, ways типа string	5
5		Инициализация указателя tek на объект класса cl_base и присвоение ему значение текущего объекта	6
6		Вывод переноса строки	7
7		Ввод значение переменной command	8
8	command равна "END"		17
			9
9	command равна "FIND"	Ввод значения переменной ways	10
			13
10		Добавляем к переменной out типа string переменной ways	11
11	Результат работы метода location_objects_way с переданным параметром ways, вызванного для указателя tek не равен nullptr	Добавляем к переменной out "Object name" добавляем результат метода get_object_name, вызванного для результата работы метода location_objects_way с переданным параметром ways, вызванного для указателя tek, добавляем переход на следующую строку	17
			12
12		Добавляем к переменной out "Object is not found"	17
13	command равна "SET"	Ввод значение переменной ways	14
			17
1	Результат работы метода	К указателю tek присваивается результат работы	15

№	Предикат	Действия	№ перехода
4	location_objects_way с переданным параметром ways, вызванного для указателя tek не равен nullptr	метода location_objects_way с переданным параметром ways вызванного для указателя tek	
			16
15		Добавляем к переменной out "Object is set" добавляем результат метода get_object_name, вызванного для результата работы метода location_objects_way с переданным параметром ways, вызванного для указателя tek, добавляем переход на следующую строку	17
16		Добавляем к переменной out "Object is not found:" добавляем результат работы метода get_object_name вызванного для указателя tek добавляем пробел добавляем значение переменной ways добавляем переход на следующую строку	7
17		Удаление последнего элемента из переменной out	18
18		Вывод значение переменной out	19
19			∅

3.4 Алгоритм метода bild_tree_objects_new2 класса cl_application

Функционал: Построение дерева объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода bild_tree_objects_new2 класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление переменных name_root_object, name_subordinate_objects, name_main_object типа string	2
2		Объявление переменной element типа int	3
3		Ввод значение переменной name_root_objects	4
4		Вызов метода set_object_name с параметром name_root_object	5
5		Ввод значение переменной name_main_object	6
6	name_main_object равен "endtree"		∅
			7
7		Ввод значение переменных name_subordinate_objects и element	8
8	Результат вызванного метода location_objects_way с переданным параметром name_main_object не равен nullptr		9
			14
9	номер класса равен 2	Создает новый объект cl_1 при помощи констуктора того же класса, в качестве параметров передается результат работы метода location_objects_way с переданным параметром name_main_object и переменная name_subordinate_objects	5
			10
1	номер класса равен 3	Создает новый объект cl_2 при помощи	5

№	Предикат	Действия	№ перехода
0		констуктора того же класса, в качестве параметров передается результат работы метода location_objects_way с переданным параметром name_main_object и переменная name_subordinate_objects	
			11
11	номер класса равен 4	Создает новый объект cl_3 при помощи констуктора того же класса, в качестве параметров передается результат работы метода location_objects_way с переданным параметром name_main_object и переменная name_subordinate_objects	5
			12
12	номер класса равен 5	Создает новый объект cl_4 при помощи констуктора того же класса, в качестве параметров передается результат работы метода location_objects_way с переданным параметром name_main_object и переменная name_subordinate_objects	5
			13
13	номер класса равен 6	Создает новый объект cl_5 при помощи констуктора того же класса, в качестве параметров передается результат работы метода location_objects_way с переданным параметром name_main_object и переменная name_subordinate_objects	5
			5
14		Вывод "Object tree"	15

№	Предикат	Действия	№ перехода
15		Вызов метода new_print	16
16		Вывод перенос строки	17
17		Вывод "The head object", вывод name_main_object, вывод "is not found"	18
18			∅

3.5 Алгоритм метода location_objects_way класса cl_base

Функционал: Получение указателя на любой объект в составе дерева иерархи объектов.

Параметры: string way.

Возвращаемое значение: Указатель на объект класса cl_base.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода location_objects_way класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя temp и присвоение к нему nullptr	2
2	Первый и второй символ равен "/"	Инициализация указателя rootObj присвоение к нему указателя на текущий объект	3
			5
3	Указатель на текущий головной объект не равен nullptr	к rootObj присваивается указатель на текущий головной объект	3
			4
4		Возврат работы метода search_objects с параметром	11

№	Предикат	Действия	№ перехода
		подстроки way, со 2-го символа для объекта по указателю rootObj	
5	way равен "/"	Присвоение к переменной temp указатель на текущий объект	6
			8
6	Указатель на текущий головной объект не равен nullptr	к temp присваивается указатель на текущий головной объект вызванного для указателя temp	7
			7
7		Возвращение указателя на текущий головной объект вызванного для указа	∅
8	way равен "."	Возвращение указателя на текущий объект	∅
			9
9	Размер строки way больше или равно 1		10
			11
10	Первый символ way равен '/'	к temp присваиваем результат метода location_objects_way с переданным параметром "/"	11
		к temp присваиваем указатель на текущий объект	11
11		Объявление переменной name типа string	12
12		Поток s_way с передаваемым значение way	13
13	Символ извлекается из входного потока и добавляется к строковому объекту, до того как встретит "/"		14
			20
1	name не равен "" (пустоте)	Инициализация переменной search тип bool и	15

№	Предикат	Действия	№ перехода
4		присвоение к ней значение false	
			13
15	i меньше размера списка объектов иерархии		16
			19
16	name равен результату работы метода get_object_name для i-нного подчиннего в списке	Присваиваем к указателю temp значение указателя на объект из контейнера с позицией i	17
			18
17		Присваиваем к переменной search значение true	18
18		Увеличиваем i на 1	15
19	Search равен false	Возврат nullptr	∅
			13
20		Возврат указателя temp	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-11.

Рисунок 1 – Блок-схема алгоритма

Рисунок 2 – Блок-схема алгоритма

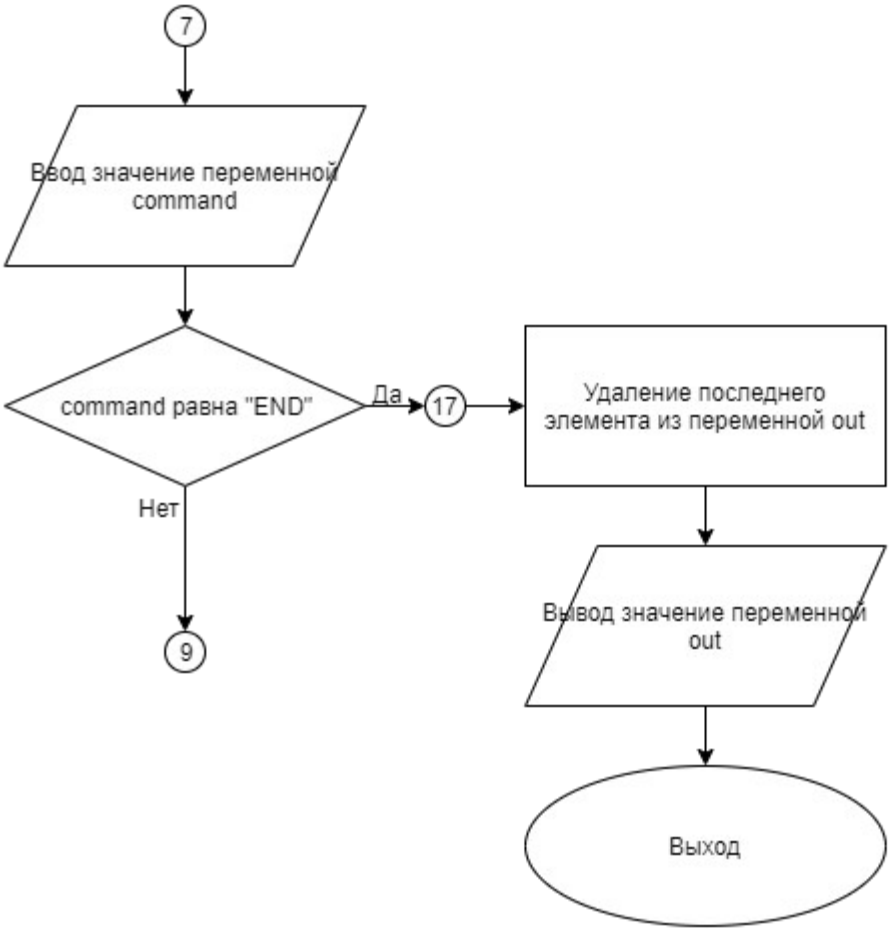

Рисунок 3 – Блок-схема алгоритма

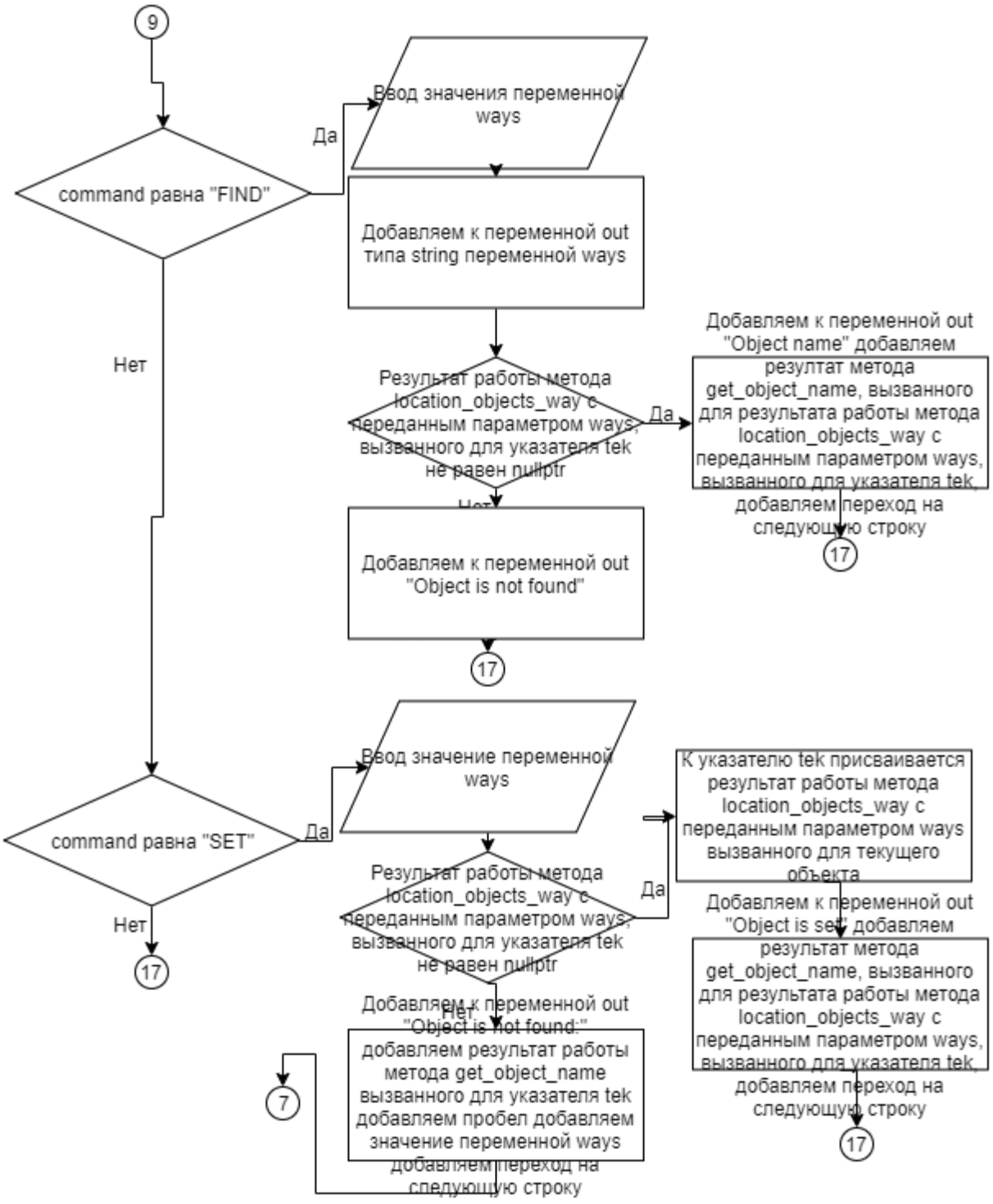

Рисунок 4 – Блок-схема алгоритма

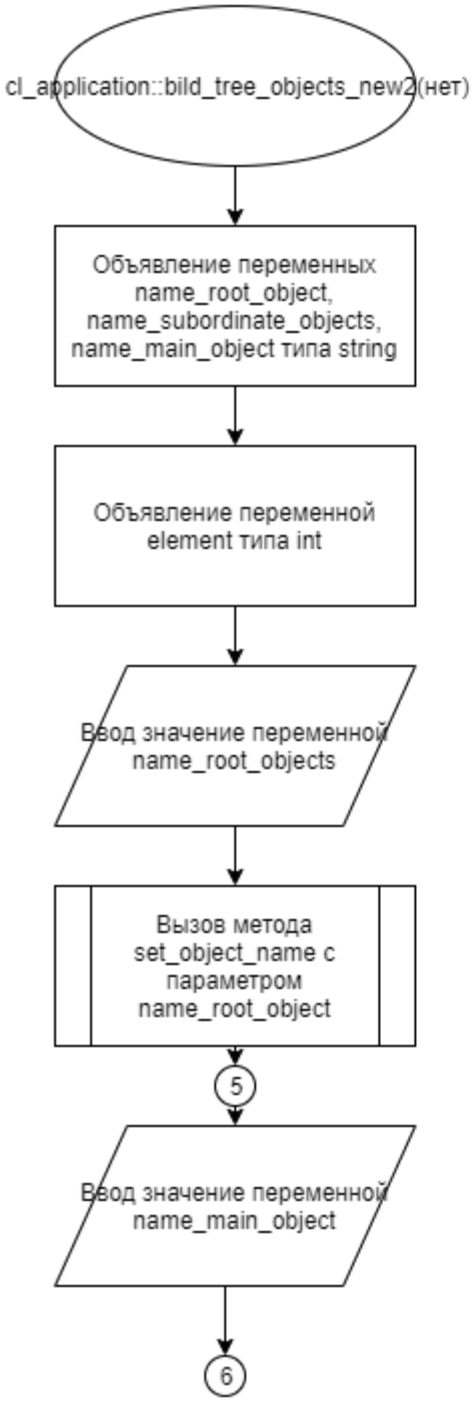

Рисунок 5 – Блок-схема алгоритма

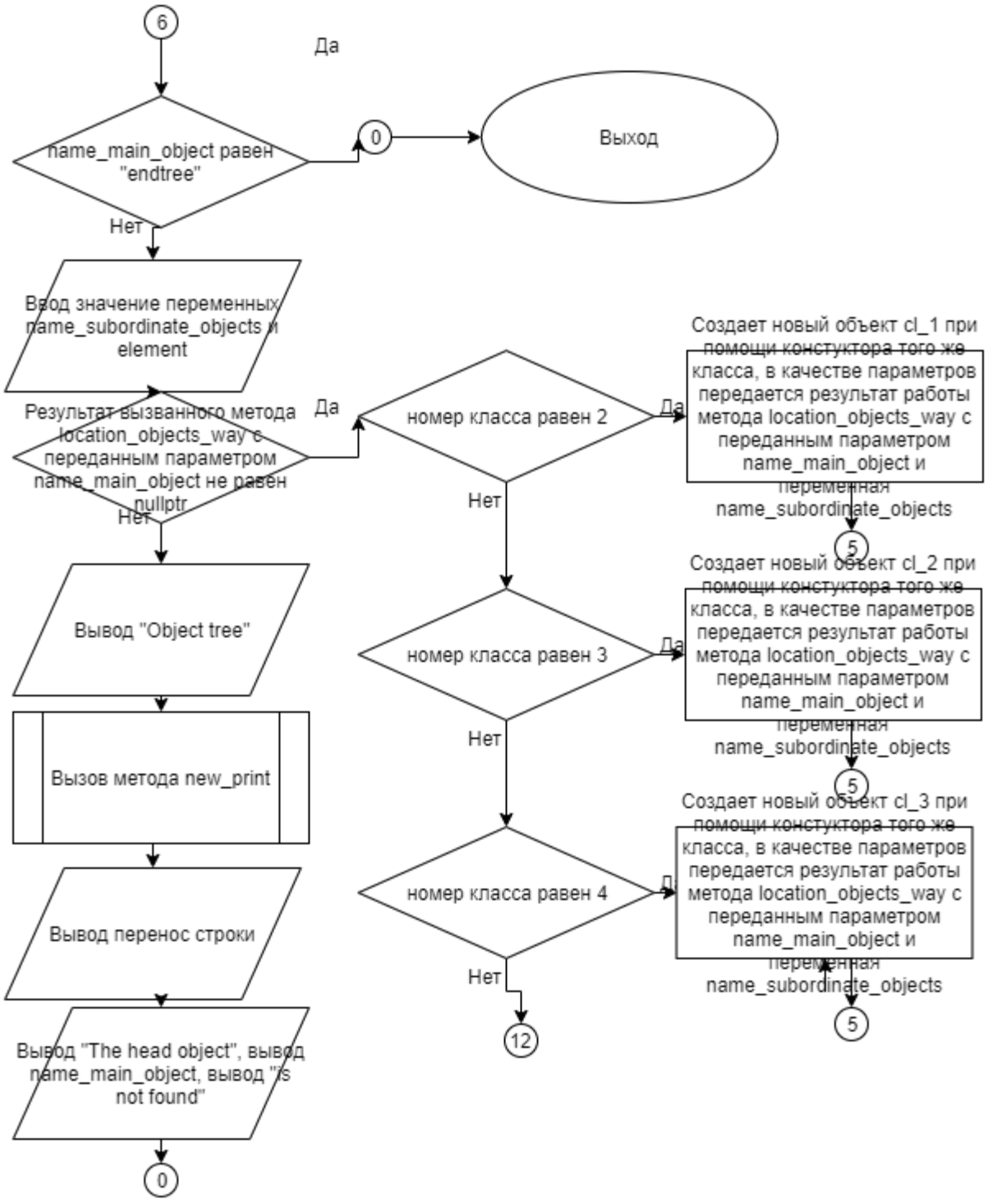

Рисунок 6 – Блок-схема алгоритма

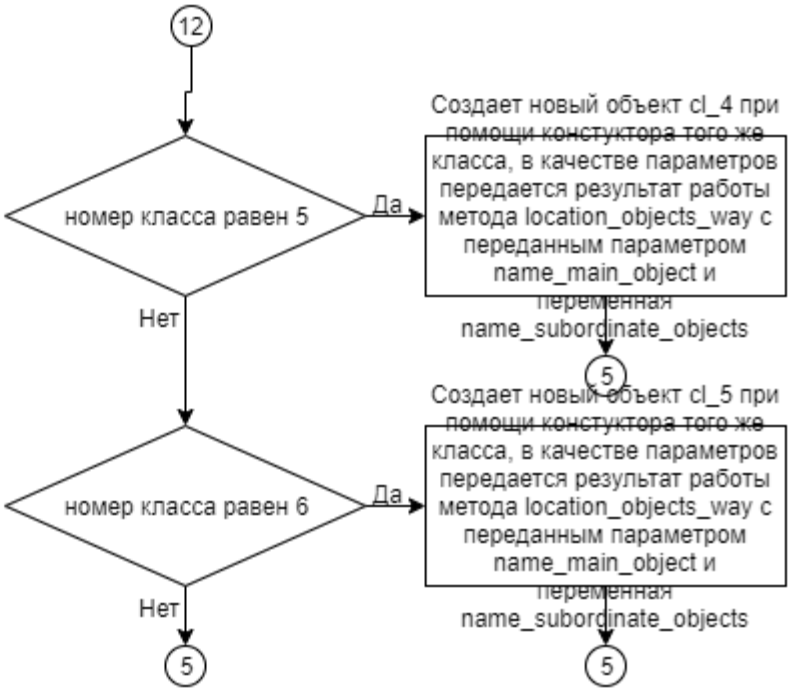

Рисунок 7 – Блок-схема алгоритма

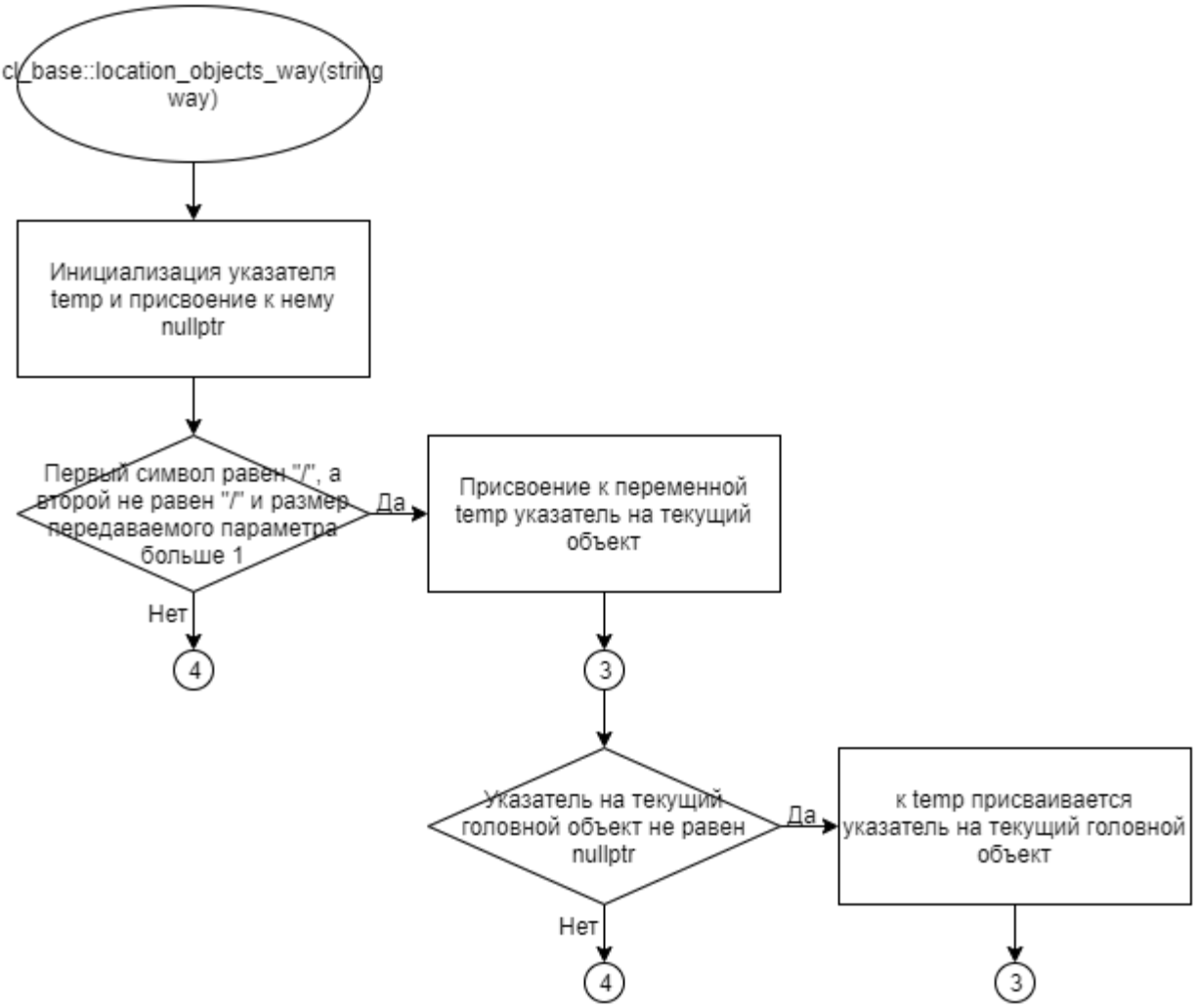

Рисунок 8 – Блок-схема алгоритма

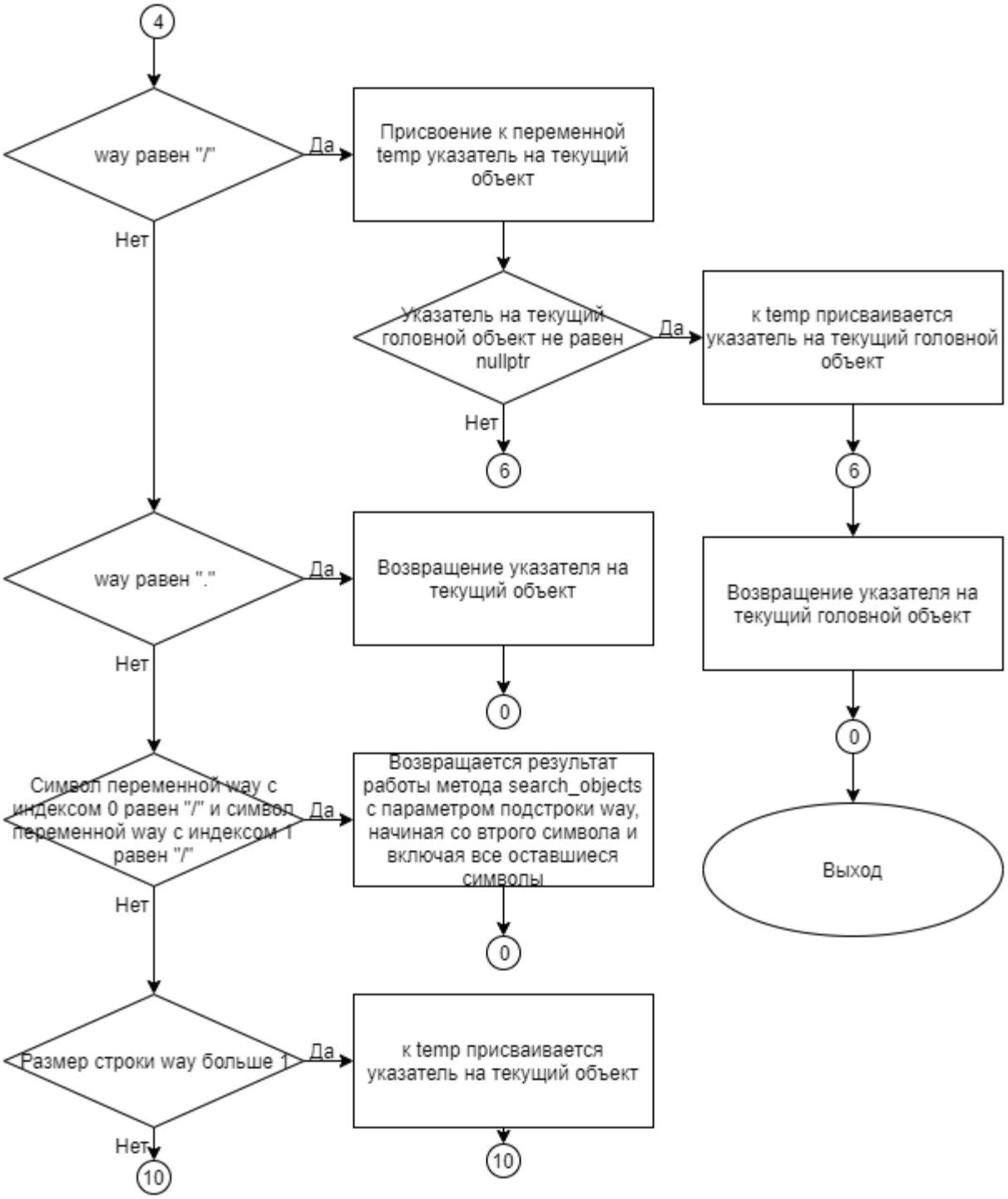

Рисунок 9 – Блок-схема алгоритма

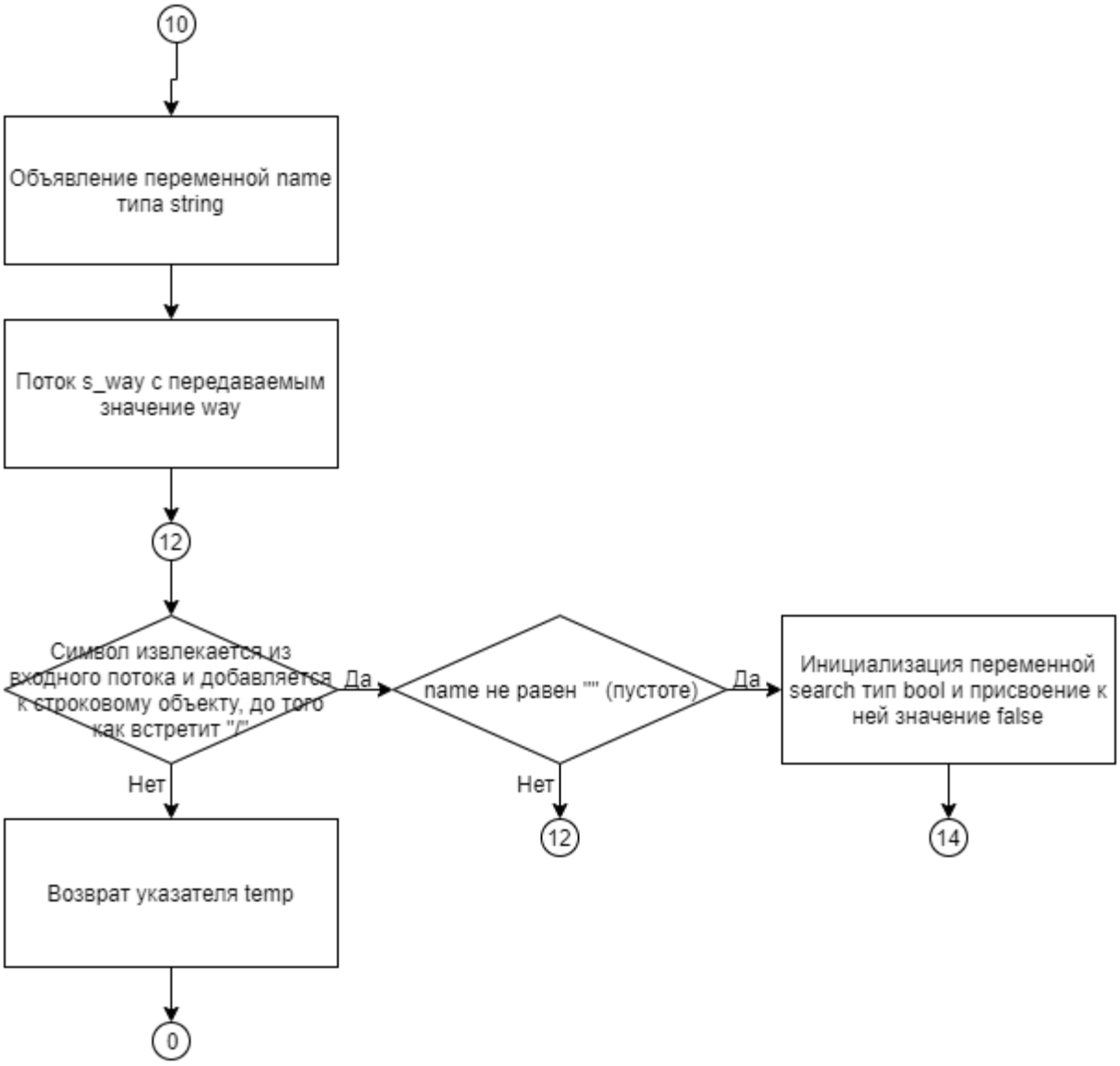

Рисунок 10 – Блок-схема алгоритма

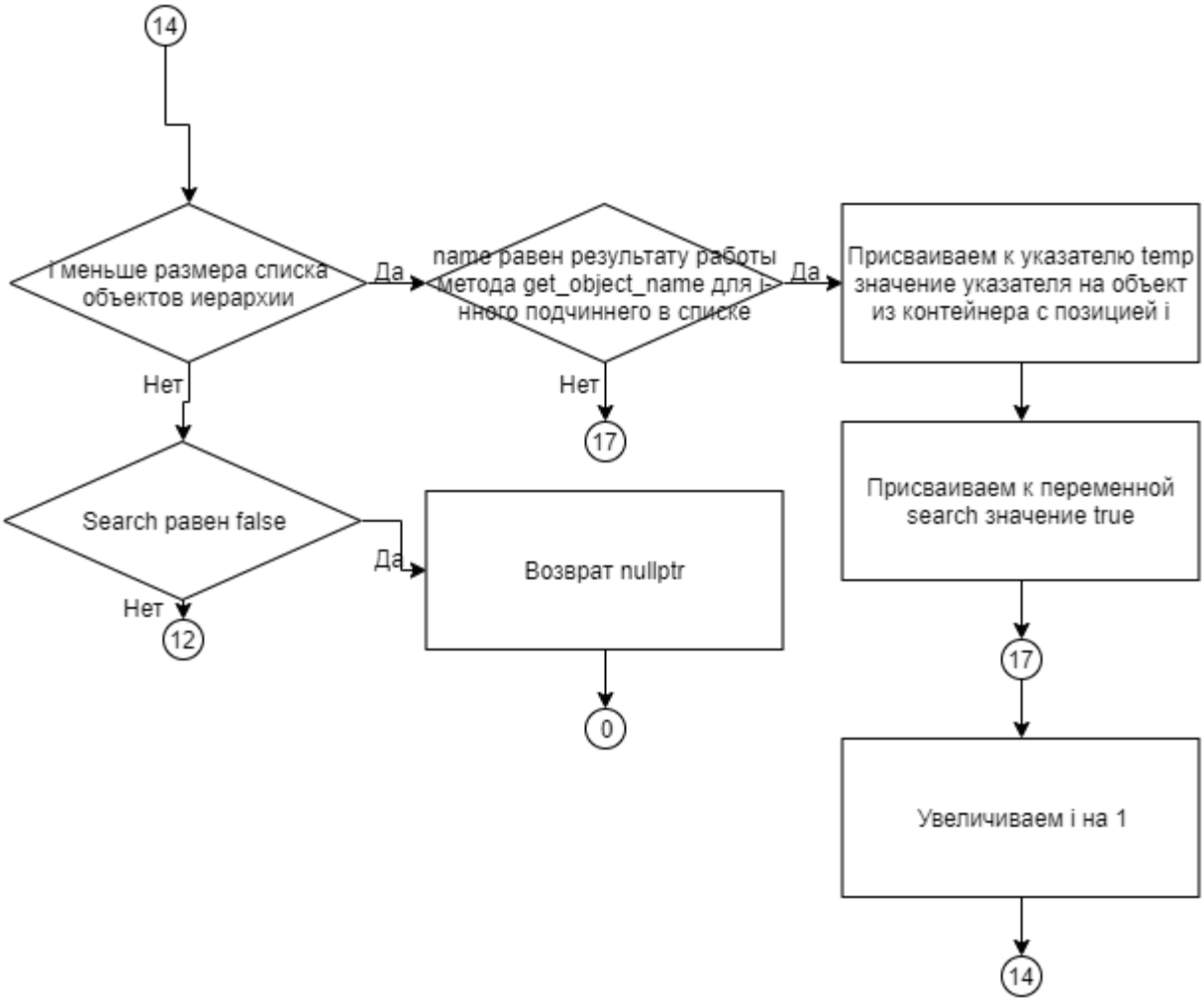

Рисунок 11 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(cl_base*                        p_head_object,                        string
s_object_name):cl_base(p_head_object,s_object_name)
{

}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef CL_1_H
#define CL_1_H
#include "cl_base.h"
class cl_1:public cl_base
{
private:

public:
cl_1(cl_base* p_head_object, string s_object_name);
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
cl_2::cl_2(cl_base*                        p_head_object,                        string
s_object_name):cl_base(p_head_object,s_object_name)
{

}
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef CL_2_H
#define CL_2_H
#include "cl_base.h"
class cl_2:public cl_base
{
private:

public:
cl_2(cl_base* p_head_object, string s_object_name);
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(cl_base*                    p_head_object,                    string
s_object_name):cl_base(p_head_object,s_object_name)
{

}
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef CL_3_H
#define CL_3_H
#include "cl_base.h"
class cl_3:public cl_base
{
private:

public:
cl_3(cl_base* p_head_object, string s_object_name);
};
#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(cl_base*                        p_head_object,                        string
s_object_name):cl_base(p_head_object,s_object_name)
{

}
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef CL_4_H
#define CL_4_H
#include "cl_base.h"
class cl_4:public cl_base
{
private:

public:
cl_4(cl_base* p_head_object, string s_object_name);
};
#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(cl_base*                        p_head_object,                        string
s_object_name):cl_base(p_head_object,s_object_name)
{

}
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef CL_5_H
```

```
#define CL_5_H
#include "cl_base.h"
class cl_5:public cl_base
{
private:

public:
cl_5(cl_base* p_head_object, string s_object_name);
};
#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
#include "cl_base.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"

cl_application::cl_application(cl_base*  p_head_object,  string  s_object_name)  :
cl_base(p_head_object, s_object_name)
{

}

/*void cl_application::bild_tree_objects()
{
	string name_root_object;
	cin >> name_root_object;
	this->set_object_name(name_root_object);
	cl_base* temp_parent = this;
	cl_base* temp_child = nullptr;
	string name_parent;
	string name_child;
	while (true)
	{
		cin>> name_parent >> name_child;
		if(name_parent == name_child)
		{
			break;
		}

		if(name_parent == temp_parent->get_object_name())
		{
			temp_child = new object(temp_parent, name_child);
		}
		else
		{
			temp_parent = temp_child;
```

```
                  temp_child = new object(temp_parent, name_child);
            }
      }
}
*/

int cl_application::exec_app()
{

            string out = "";
            cout << "Object tree";//<< endl;
            new_print();
            string command, ways;
            cl_base* tek = this;


            cout << "\n";
            //command != "END"

            while(true)
            {
                  cin >> command;
                  if(command == "END"){break;}

                  else if(command == "FIND")
                  {

                        cin >> ways;
                        out += ways;
                        if(tek->location_objects_way(ways)!= nullptr)
                        {
                              out   +=   "             Object   name:   "   +   tek-
>location_objects_way(ways)->get_object_name() + "\n";
                        }
                        else
                        {
                              out += "     Object is not found\n";
                        }
                  }
                  else if(command == "SET")
                  {
                        cin >> ways;

                        if(tek->location_objects_way(ways)!= nullptr)
                        {
                              tek = tek-> location_objects_way(ways);
                              out += "Object is set: " + tek->get_object_name() +
"\n";


                        }
                        else
                        {
                              out   +=   "Object   is   not   found:   "   +   tek-
>get_object_name() + " " + ways + "\n";
                        }
                  }
```

```
            }

            out.pop_back();
            cout << out;



      return 0;
}

/*void cl_application::bild_tree_objects()
{
      int element;
      string name_root_object, name_subordinate_objects,name_main_object;
      cin >> name_root_object;
      this->set_object_name(name_root_object);
      cl_base* temp_parent = this;
      cl_base* temp_child = nullptr;
      string name_parent;
      string name_child;
      while (true)
      {
            cin>> name_main_object;
            if(name_main_object == "endtree")
            {
                  break;
            }
            cin >> name_subordinate_objects >> element;
            cl_base* tort = search_objects(name_main_object);
            if(element == 2)
            {
                  new cl_1(tort, name_subordinate_objects);
            }
            else if(element == 3)
            {
                  new cl_2(tort, name_subordinate_objects);
            }
            else if(element == 4)
            {
                  new cl_3(tort, name_subordinate_objects);
            }
            else if(element == 5)
            {
                  new cl_4(tort, name_subordinate_objects);
            }
            else if(element == 6)
            {
                  new cl_5(tort, name_subordinate_objects);
            }

      }
      string stroka;
      int chislo;
      cl_base* temp;

      while(cin >> stroka >> chislo)
      {
```

```
            search_objects(stroka)->set_state(chislo);
      }
}
*/

void cl_application::bild_tree_objects_new2()
{
      string name_root_object, name_subordinate_objects, name_main_object;
      int element;
      cin>>name_root_object;
      set_object_name(name_root_object);

      while(true)
      {
            cin>> name_main_object;
            if(name_main_object == "endtree")
            {
                  break;
            }

            cin >> name_subordinate_objects >> element;

            if(location_objects_way(name_main_object) != nullptr)
            {
                  if(element == 2)
                  {
                        new           cl_1(location_objects_way(name_main_object),
name_subordinate_objects);

                  }
                  else if(element == 3)
                  {
                        new           cl_2(location_objects_way(name_main_object),
name_subordinate_objects);

                  }
                  else if(element == 4)
                  {
                        new           cl_3(location_objects_way(name_main_object),
name_subordinate_objects);

                  }
                  else if(element == 5)
                  {
                        new           cl_4(location_objects_way(name_main_object),
name_subordinate_objects);

                  }
                  else if(element == 6)
                  {
                        new           cl_5(location_objects_way(name_main_object),
name_subordinate_objects);

                  }
            }

            else
```

```
            {
                  //success = "Дерево не построено";
                  cout<<"Object tree";
                  new_print();
                  cout<<endl;
                  cout<<"The head object "<< name_main_object<<" is not found";


                  exit(0);
            }
      }
      //success = "Дерево построено";
}
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H
#include "cl_base.h"
#include "object.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
class cl_application:public cl_base
{
private:
      string success;
public:
      cl_application(cl_base*     p_head_object,     string     s_object_name     =
"Base_object");
      void bild_tree_objects();
      void bild_tree_objects_new();
      int exec_app();
      void bild_tree_objects_new2();

};
#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
cl_base::cl_base(cl_base * p_head_object, string s_object_name)
{
```

```
	this ->  s_object_name = s_object_name;
	this ->  p_head_object =  p_head_object;
	if (p_head_object != nullptr)
	{
		p_head_object -> subordinate_objects.push_back(this);
	}

}

cl_base::~cl_base()
{
	for (int i = 0; i <subordinate_objects.size();i++)
	{
		delete subordinate_objects[i];
	}
}

void cl_base::change_head_object(cl_base * p_head_object)
{
	if(p_head_object != nullptr && this->p_head_object != nullptr)
	{
		cl_base* temp = this->p_head_object;
		p_head_object->subordinate_objects.push_back(this);
		this->p_head_object = p_head_object;

		for(int i= 0; i< temp->subordinate_objects.size();i++)
		{
			if(this == temp->subordinate_objects[i])
			{
				temp->subordinate_objects.erase(temp-
>subordinate_objects.begin()+i);
				break;
			}
		}
	}
}

void cl_base::set_object_name(std::string s_object_name)
{
	this->s_object_name = s_object_name;
}

// cl_base *  - это указатель на базовый класс
cl_base * cl_base::get_head_object()
{
	return  p_head_object;
}

string cl_base::get_object_name()
{
	return s_object_name;
}

void cl_base::print()
{
	if(this->subordinate_objects.size()!=0)
	{
```

```
            cout<<endl<<this->s_object_name;
            for(int i = 0; i < this->subordinate_objects.size();i++)
            {
                  cout<<"  "<< this->subordinate_objects[i]->s_object_name;

            }
                  //cout<<endl;
      }

      for(int i = 0; i < this->subordinate_objects.size();i++)
      {
            this->subordinate_objects[i]->print();
      }
}

/*void cl_base::set_state(int i_state)
{
      if(i_state != 0)
      {
            cl_base* pocco = p_head_object;
            while(pocco != nullptr)
            {
                  if(pocco -> state == 0)
                  {
                        return;
                  }
                  pocco = pocco->p_head_object;

            }
            state = i_state;
      }
      else
      {
            state = i_state;
            for(int i = 0; i < subordinate_objects.size(); i++)
            {
                  subordinate_objects[i]->set_state(0);
            }

      }

}
*/
/*cl_base* cl_base::search_objects(string tort)
{
      if(s_object_name == tort)
      {
            return this;
      }
      else
      {
            for(int i = 0; i < subordinate_objects.size(); i++)
            {
                  if(subordinate_objects[i] -> s_object_name == tort)
                  {
                        return subordinate_objects[i];
```

```
                  }
            }

            for(int i = 0; i < subordinate_objects.size();i++)
            {
                  cl_base* temp = subordinate_objects[i]-> search_objects(tort);
                  if(temp != nullptr)
                  {
                        return temp;
                  }
            }
      }
      return nullptr;
}*/
cl_base* cl_base::search_objects(string s_object_name)
{
      if(this->get_object_name() == s_object_name)
      {
            return this;
      }
      else if(this->subordinate_objects.size()>0)
      {
            for(int i = 0; i < subordinate_objects.size(); i++)
            {
                  /*if(subordinate_objects[i] -> s_object_name == tort)
                  {
                        return subordinate_objects[i];
                  }*/
                  cl_base*       namej       =       this->subordinate_objects[i]->
search_objects(s_object_name);
                  if(namej != nullptr)
                  {return namej;}
            }


      }
      return nullptr;
}


void cl_base::new_print()
{
      int dip = -1;
      cl_base * temp = this;
      while(temp != nullptr)
      {
            temp = temp -> p_head_object;
            dip++;
      }
      cout << endl;
      for(int i = 0; i < dip; i++)
      {
            cout<<"    ";
      }
      cout << s_object_name;
      for(int i = 0; i < subordinate_objects.size(); i++)
      {
```

```
            subordinate_objects[i]->new_print();
      }

}

void cl_base::readlines()
{

      int dip = -1;
      cl_base* temp = this;
      while(temp != nullptr)
      {
            temp = temp -> p_head_object;
            dip++;
      }

      /*for(int i =0; i < dip; i++)
      {
            cout<<"    ";
      }
      cout<<s_object_name << " " << ((state  == 0) ? "is not ready": "is ready") ;


      for(int i = 0; i < subordinate_objects.size(); i++)
      {
            cout<<endl;
            subordinate_objects[i] -> readlines();
      }*/



}

/*cl_base* cl_base::location_objects_way(string way)
{
      cl_base* temp = nullptr;

      if (way[0] == '/' && way[1] == '/')
      {
            cl_base* rootObj = this;
            while(rootObj-> p_head_object != nullptr)
            {
                  rootObj = rootObj -> p_head_object;
            }
            return rootObj->search_objects(way.substr(2));
      }
      else if (way == "/")
      {
            temp = this;
            while(temp-> p_head_object != nullptr)
            {
                  temp = temp -> p_head_object;
            }
            return temp;
      }
      else if (way == ".")
      {
```

```
            return this;
      }
      else if (way.size() >= 1)
      {
            if(way[0] == '/')
            {
                  temp = location_objects_way("/");
            }
            else
            {
            temp = this;
            }
      }

      string name;
      stringstream s_way(way);

      while(getline(s_way, name, '/'))
      {

            if (name != "")
            {
                  bool search = false;
                  for(int i = 0; i < temp->subordinate_objects.size(); i++)
                  {
                        if       (name       ==       temp->subordinate_objects[i]-
>get_object_name())
                        {
                              temp = temp->subordinate_objects[i];
                              search = true;
                              break;
                        }
                  }
                  if (search == false)
                  {
                        return nullptr;
                  }
            }
      }

      return temp;
}*/

cl_base* cl_base::location_objects_way(string way)
{
      cl_base* temp = nullptr;


      if (way == "/")
      {
            temp = this;
            while(temp-> p_head_object != nullptr)
            {
                  temp = temp -> p_head_object;
            }
            return temp;
      }
```

```
      else if (way == ".")
      {
            return this;
      }
      else if (way[0] == '/' && way[1] == '/')
      {
            cl_base* rootObj = this;
            while(rootObj-> get_head_object() != nullptr)
            {
                  rootObj = rootObj -> get_head_object();
            }
            return rootObj->search_objects(way.substr(2));
      }
      else if (way.size() >= 1)
      {
            if(way[0] == '/')
            {
                  temp = location_objects_way("/");
            }
      else
      {
            temp = this;
      }
      }

      string name;
      stringstream s_way(way);

      while(getline(s_way, name, '/'))
      {

            if (name != "")
            {
                  bool search = false;
                  for(int i = 0; i < temp->subordinate_objects.size(); i++)
                  {
                        if       (name       ==       temp->subordinate_objects[i]-
>get_object_name())
                        {
                              temp = temp->subordinate_objects[i];
                              search = true;
                              break;
                        }
                  }
                  if (search == false)
                  {
                        return nullptr;
                  }
            }
      }

      return temp;
}
```

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 <sstream>
using namespace std;
class cl_base
{
private:
      string s_object_name;                          //наименование объекта
      cl_base* p_head_object;                       //указатель на головной объект
      vector < cl_base * > subordinate_objects;       //указатели  на  подчиненные
объекты
      int state = 0;                                                    //Поля
отвечающее за хранения состояния текущего объекта
public:
      cl_base(cl_base * p_head_object, string s_object_name = "Base_object");
      ~cl_base();
      void set_object_name(string s_object_name);//
      //определение наименования объекта
      string get_object_name();//
      //наименование объекта
      void change_head_object( cl_base*);
      //переопределение головного объекта
      cl_base * get_head_object();//            //указатель на головной объект
      void print();
      void set_state(int i_state);
      //cl_base * search_objects(string tort);
      cl_base * search_objects(string s_object_name);
      void new_print();
      void readlines();
      cl_base* location_objects_way(string way);
};

#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);
      ob_cl_application.bild_tree_objects_new2();  // построение дерева объектов
      return ob_cl_application.exec_app();   // запуск системы
}
```

5.16 Файл object.cpp

Листинг 16 – object.cpp

```
#include "object.h"
object::object(cl_base*                    p_head_object,                    string
s_object_name):cl_base(p_head_object,s_object_name)
{

}
```

5.17 Файл object.h

Листинг 17 – object.h

```
#ifndef OBJECT_H
#define OBJECT_H
#include "cl_base.h"
class object:public cl_base
{
private:

public:
object(cl_base* p_head_object, string s_object_name);
};
#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 5.

Таблица 5 – 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_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	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	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

ЗАКЛЮЧЕНИЕ

В ходе изучения ООП я узнал о таких новых понятиях как:

Наследование - свойства системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью. Класс, от которого производится наследование называется базовыми, родительским или суперклассом. Новый класс - потомком, наследником, дочерним или производным классом.

Инкапсуляция - механизм программирования, который связывает код и данные, которыми он манипулирует, и при этом предохраняет их от вмешательства извне и неправильного использования.

Вся разработка проводилась в среде АСО Avrora, она мне понравилась своей удобней интерфейсом, большими количествами тем, подходом к сдаче работ, не нужно каждый раз подходить к преподавателю, а просто отправить дистанционно, но везде есть свои недочёты, например, в Авроре это отсутствие процедуры вставки. Процедура вставка намного упростила бы работу программиста.

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».

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).