СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Формирование и работа с иерархией объектов программы-системы.

Создание объектов и построение исходного иерархического дерева объектов.

Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность).

Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно.

Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main).

Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных программах динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей,

существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны.

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

Расширить функциональность базового класса:

- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr);
- метод вывода дерева иерархии объектов;
- метод вывода дерева иерархии объектов и отметок их готовности;
- метод установки готовности объекта реализовать (доработать) следующим образом.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется.

При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

1. Построить дерево объектов системы (в методе коневого объекта построения исходного дерева объектов).

2. В методе корневого объекта запуска моделируемой системы реализовать:

2.1. Вывод на консоль иерархического дерева объектов в следующем виде:

root

ob_1

ob_2

ob_3

ob_4

ob_5

ob_6

ob_7

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам).

2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

root is ready

ob_1 is ready

ob_2 is ready

ob_3 is ready

ob_4 is not ready

ob_5 is not ready

ob_6 is ready

ob_7 is not ready

1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии.

Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка

«Наименование корневого объекта»

Со второй строки

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

. . . . .

endtree

Со следующей строки вводятся команды включения или отключения объектов

«Наименование объекта» «Номер состояния объекта»

. . . . .

Признаком завершения ввода является конец потока входных данных.

Пример ввода

app_root

app_root object_01 3

app_root object_02 2

object_02 object_04 3

object_02 object_05 5

object_01 object_07 2

endtree

app_root 1

object_07 3

object_01 1

object_02 -2

object_04 1

1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

Object tree

«Наименование корневого объекта»

«Наименование объекта 1»

«Наименование объекта 2»

«Наименование объекта 3»

. . . . .

The tree of objects and their readiness

«Наименование корневого объекта» «Отметка готовности»

«Наименование объекта 1» «Отметка готовности»

«Наименование объекта 2» «Отметка готовности»

«Наименование объекта 3» «Отметка готовности»

. . . . .

«Отметка готовности» - равно «is ready» или «is not ready»

Отступ каждого уровня иерархии 4 позиции.

Пример вывода

```
Object tree
app_root
    object_01
        object_07
    object_02
        object_04
        object_05
The tree of objects and their readiness
app_root is ready
    object_01 is ready
        object_07 is not ready
    object_02 is ready
        object_04 is ready
        object_05 is not ready
```

2 МЕТОД РЕШЕНИЯ

Для решения этой задачи требуется:

Объекты стандартного потока ввода/вывода cin/cout

условный оператор if

оператор цикла с предусловием while

объекты классов cl_1, cl_2, cl_3, cl_4, cl_5, количество и имена которых определяется пользователем

1. Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс в иерархии классов. Содержит основные поля и методы.	
		cl_1	public		2
		cl_2	public		3
		cl_3	public		4
		cl_4	public		5
		cl_5	public		6

2	cl_1			Произвольный класс для создания объекта	
3	cl_2			Произвольный класс для создания объекта	
4	cl_3			Произвольный класс для создания объекта	
5	cl_4			Произвольный класс для создания объекта	
6	cl_5			Произвольный класс для создания объекта	

2. Класс cl_base:

- Свойства/поля:

1. Поля отвечающее за хранения состояния текущего объекта
    1. Наименование - state;
    2. Тип - целочисленный;
    3. Модификатор доступа - закрытый;

- Методы:

Метод set_state:

Функционал - метод установки готовности объекта;

Метод search_objects:

Функционал - метод поиска объекта на дереве иерархии по имени;

Метод new_print:

Функционал - метод вывода дерева иерархии объектов;

Метод readlines:

Функционал - метод вывода дерева иерархии объектов и отметок их готовности;

3. Класс cl_application:

Свойства/поля:

Методы:

Метод bild_tree_objects:

Функционал - метод для построение дерева иерархии объектов

Метод exec_app:

Функционал - метод для запуска системы

3 ОПИСАНИЕ АЛГОРИТМА

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

3.1. 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 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создаем объект ob_cl_application класса cl_application посредством параметризированного конструктора с параметром nullptr	2
2		вызов метода beel_tree_objects объекта ob_cl_application	3
3		Вызов метода exec_app объекта ob_cl_application	4
4		Возвращается результат работы метода exec_app	∅

3.2. Алгоритм метода beel_tree_objects класса cl_application

Функционал: метод построения исходного дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода beel_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление переменной element целочисленного типа	2
2		Объявление переменных строкового типа name_root_object, name_subordinate_objects, name_main_object	3
3		Ввод с клавиатуры значения переменной name_root_objects	4
4		Вызов метода set_objects_name с параметром name_root_object для текущего объекта	5
5		Объявление указателя на объект базового класса temp_parent и присвоение к нему указатель на текущий объект	6
6		Объявление указателя на объект базового класса temp_child и присвоение к нему nullptr	7
7		Объявление переменыых строкового типа name_parent и name_child	8
8		Ввод с клавиатуры переменную name_main_objects	9
9	имя головного объекта равен строке endtree		16
		Ввод с клавиатуры переменных name_subordinate_objects и element	10
10		Объявление указателя tort на объект базового класса и присвоение к нему результата метода search_objects с параметром name_main_objects	11
11	номер класса принадлежностии объекта равен 2	Создаем новый объект класса cl_1 при помощи конструктора того же класса, в качестве параметров передается указатель tort и переменная name_subordinate_objects	8

№	Предикат	Действия	№ перехода
			12
12	номер класса принадлежностии объекта равен 3	Создаем новый объект класса cl_2 при помощи конструктора того же класса, в качестве параметров передается указатель tort и переменная name_subordinate_objects	8
			13
13	номер класса принадлежностии объекта равен 4	Создаем новый объект класса cl_3 при помощи конструктора того же класса, в качестве параметров передается указатель tort и переменная name_subordinate_objects	8
			14
14	номер класса принадлежностии объекта равен 5	Создаем новый объект класса cl_4 при помощи конструктора того же класса, в качестве параметров передается указатель tort и переменная name_subordinate_objects	8
			15
15	номер класса принадлежностии объекта равен 6	Создаем новый объект класса cl_5 при помощи конструктора того же класса, в качестве параметров передается указатель tort и переменная name_subordinate_objects	8
			8
16		Объявление переменной строкового типа string	17
17		Объявление переменной целочисленного типа chislo	18
18		Объявление указателя temp на объект базового класса	19
19	Ввод с клавиатуры значения переменных stroka и chislo до того как данные	Поиск объектов при помощи вызванного метода search_objects с переданным параметром stroka, после нахождения объекта присваиваем ему	19

№	Предикат	Действия	№ перехода
	закончатся	готовность или не готовность с помощью вызванного метода set_state с переданным параметром chislo	
			∅

3.3. Алгоритм метода set_state класса cl_base

Функционал: метод установки готовности объекта.

Параметры: целочисленное, i_state - состояние текущего объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_state класса cl_base

№	Предикат	Действия	№ перехода
1	Переданный параметр i_state не равен нулю	Объявление указателя россо на объект базового класса и присвоение к нему указателя на головной объект	2
			5
2	указатель россо на объект базового класса не равен nullptr		3
			4
3	Поля state у объекта на который указывает указатель россо равен нулю		∅
		к россо присваивается указатель на головной объект	2
4		Присваиваем к полю state значение переменной i_state у текущего объекта	∅

№	Предикат	Действия	№ перехода
5		Присваиваем к полю state значение переменной i_state у текущего объекта	6
6	i меньше размера списка объектов	Вызываем метод set_state с передаваемым значением нуль для i-нного подчиненного	7
			∅
7		Увеличичваем i на 1	6

3.4. Алгоритм метода search_objects класса cl_base

Функционал: метод поиска объекта на дереве иерархии по имени.

Параметры: строковой tort - имя искомового объекта.

Возвращаемое значение: метод возвращает указатель на найденный объект или nullptr.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода search_objects класса cl_base

№	Предикат	Действия	№ перехода
1	поля s_object_name равен значению переданного параметра tort	Возвращает указатель на текущий объект	∅
			2
2	i меньше размера списка объектов		3
			5
3	У i-ого подчиннего (из вектора subordinate_objects для текущего объекта) поле s_object_name равно переданному параметру tort	Возврат i-ный подчиненный вектора subordinate_objects для текущего объекта	∅

№	Предикат	Действия	№ перехода
			4
4		Увеличиваем i на 1	2
5	i меньше размера списка объектов	Объявляем указатель на объект базового класса temp и присваиваем к нему результат вызова метода search_objects с параметром tort вызванного для i-нового подчиненного	6
			8
6	Значение переменной temp не равен нулю	Возврат значения переменной temp	∅
			7
7		Увеличиваем i на 1	5
8		Возврат nullptr	∅

3.5. Алгоритм метода new_print класса cl_base

Функционал: метод вывода дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода new_print класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация переменной dip и присвоение к ней -1	2
2		Объявление указателя на объект базового класса и присвоение к нему указателя на текущий объект	3
3	Указатель на текущий объект существует	Присвоение к temp указатель на головной объект	4
			5

№	Предикат	Действия	№ перехода
4		Увеличиваем dip на 1	3
5	i меньше dip	Вывод 4 пробела	6
			7
6		Увеличиваем i на 1	5
7		Вывод на экран имя текущего объекта и переход на следующую строку	8
8		Переход на следующую строку	9
9	i меньше размера списка объектов	Вызов метода new_print для i-нного подчиненного	10
			∅
10		Увеличиваем i на 1	9

3.6. Алгоритм метода readlines класса cl_base

Функционал: метод вывода дерева иерархии объектов и отметок их готовности.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода readlines класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация переменной dip и присвоение к ней -1	2
2		Объявление указателя на объект базового класса и присвоение к нему указателя на текущий объект	3
3	Указатель на текущий объект существует	Присвоение к temp указатель на головной объект	4

№	Предикат	Действия	№ перехода
			5
4		Увеличиваем dip на 1	3
5	i меньше dip	Вывод 4 пробела	6
			6
6		увеличиваем i на 1	7
7		Вывод на экран поля s_object name и символ пробела	8
8	поля state у текущего объекта равен нулю	Вывод is not ready	9
		Вывод is ready	9
9	i меньше размера списка объектов	Переход на следующую строку	10
			∅
10		Вызов метод readlines для i-нного подчиннего	11
11		Увеличиваем i на 1	9

3.7. Алгоритм метода exec_app класса cl_application

Функционал: метод запуска приложения.

Параметры: нет.

Возвращаемое значение: нуль или код ошибки.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод на экран "Object tree" и переход на следующую строку	2
2		Вызов метода new_print	3

№	Предикат	Действия	№ перехода
3		Вывод на экран "The tree of objects and their readlines	4
4		Вызов метода readlines	∅

4 БЛОК-СХЕМА АЛГОРИТМА

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

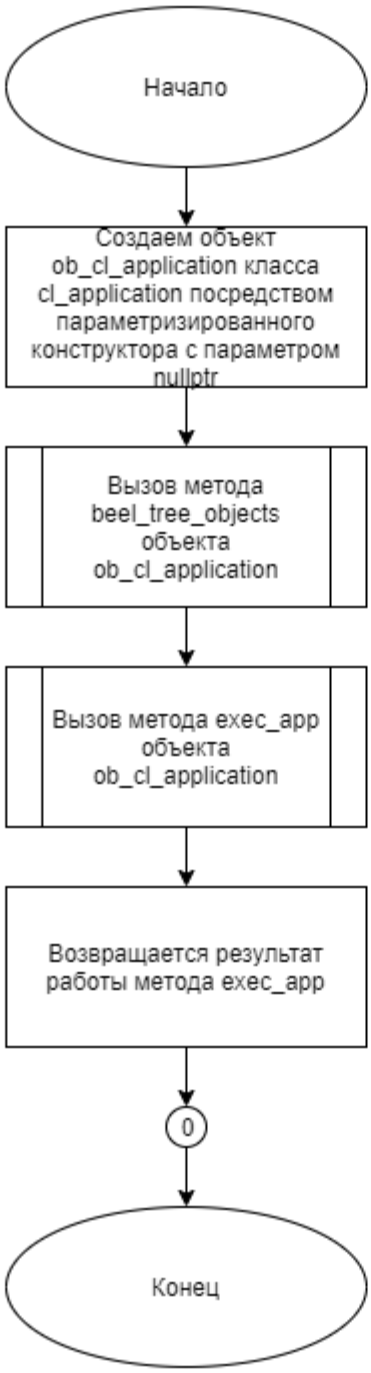

Рисунок 1 – Блок-схема алгоритма

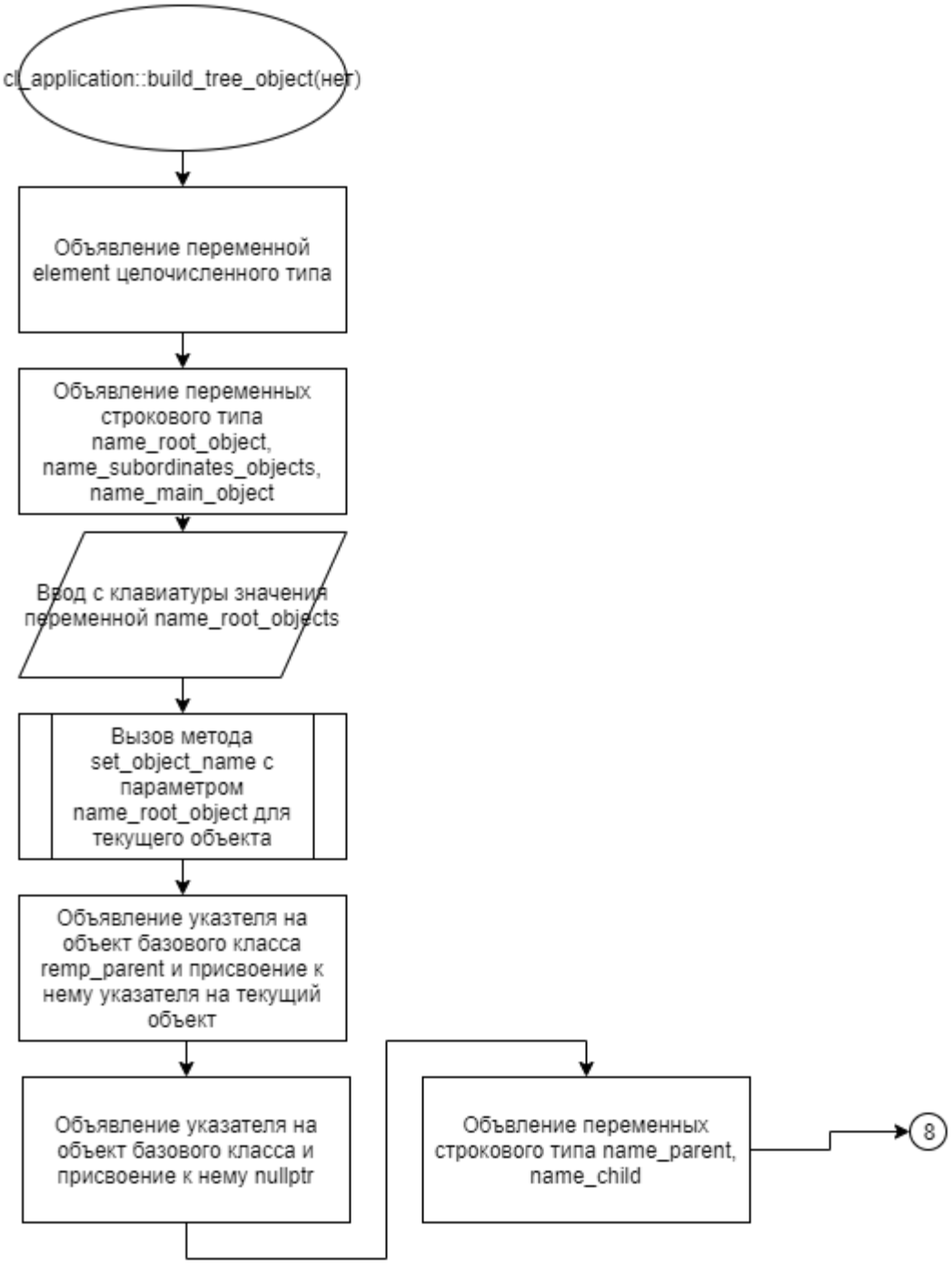

Рисунок 2 – Блок-схема алгоритма

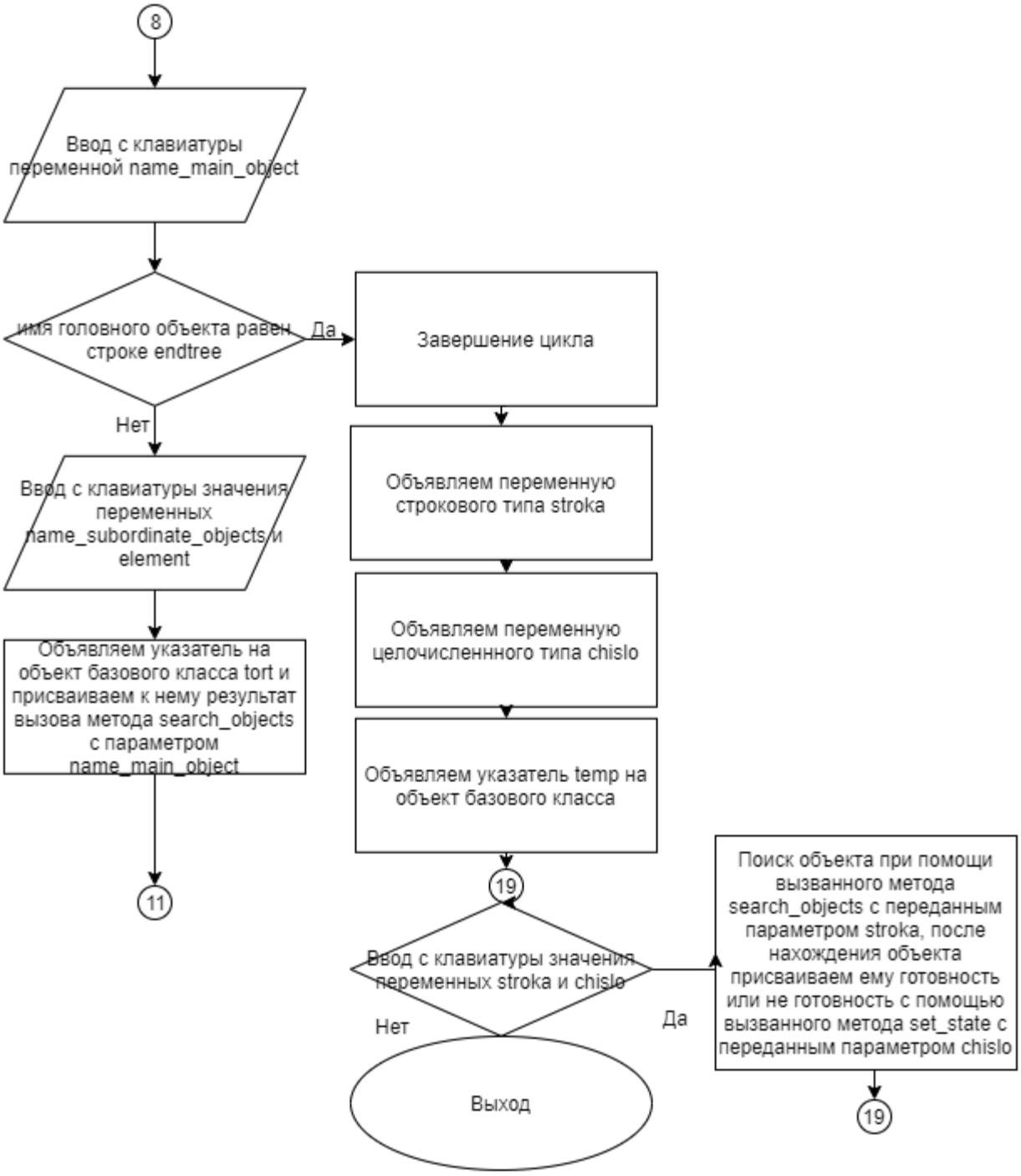

Рисунок 3 – Блок-схема алгоритма

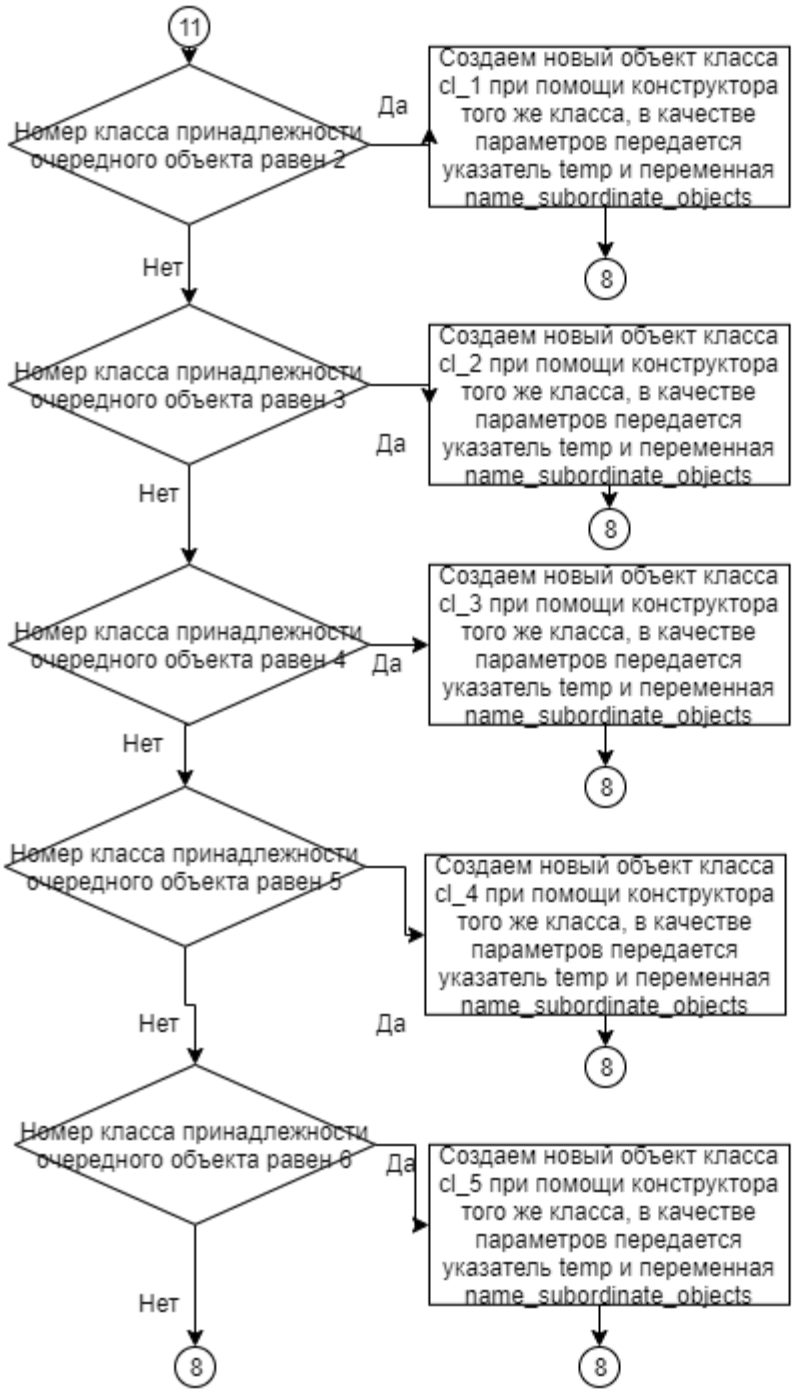

Рисунок 4 – Блок-схема алгоритма

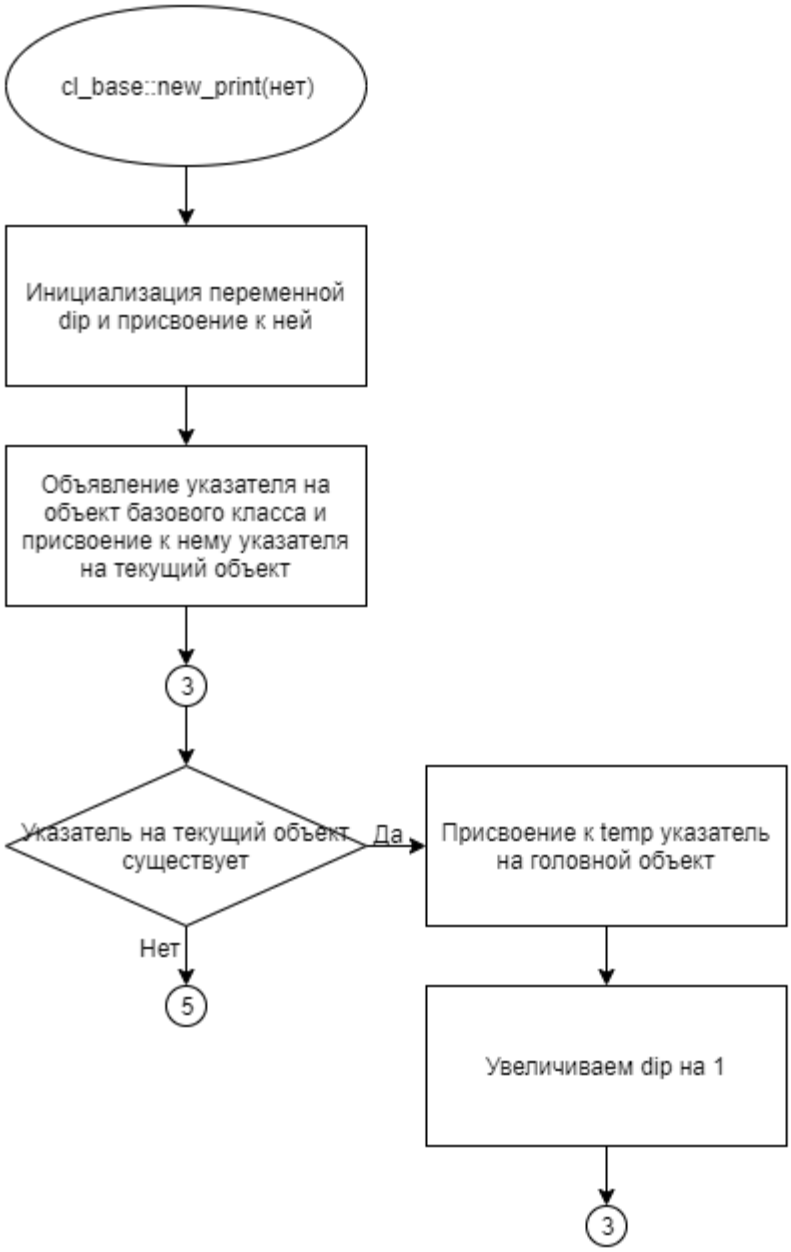

Рисунок 5 – Блок-схема алгоритма

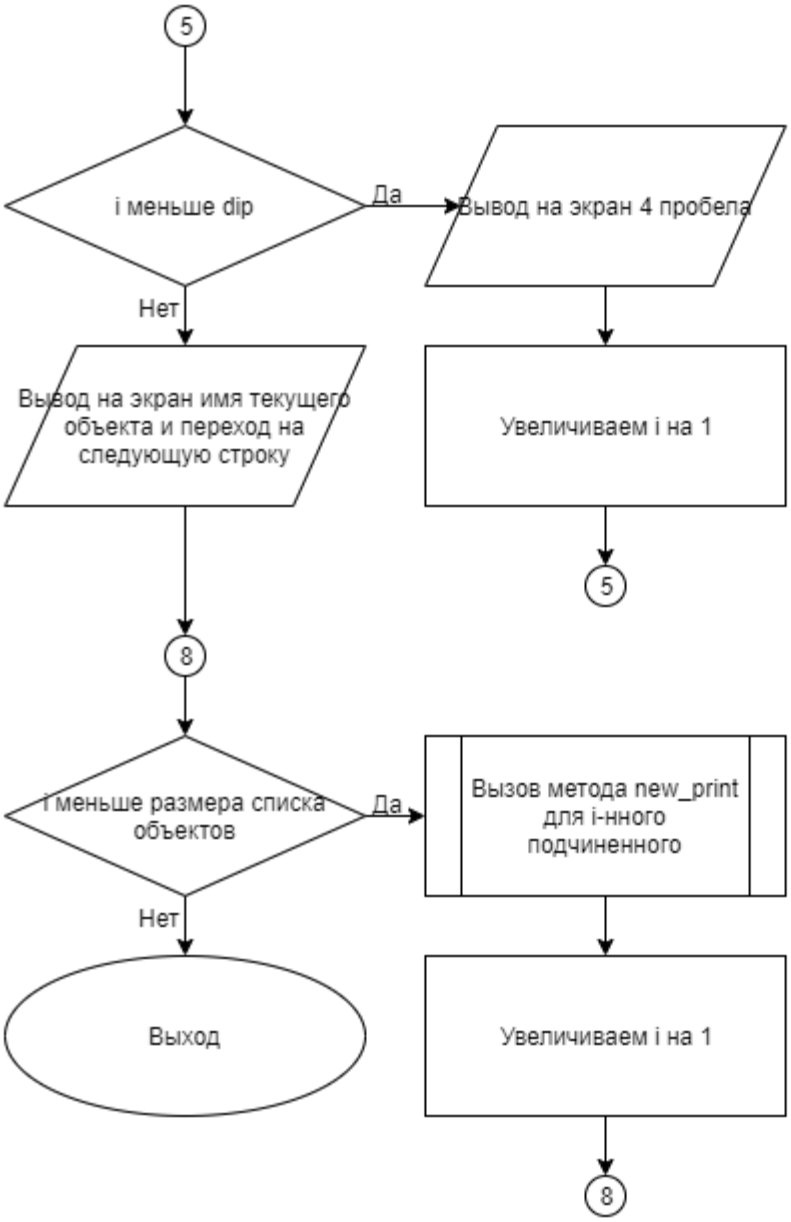

Рисунок 6 – Блок-схема алгоритма

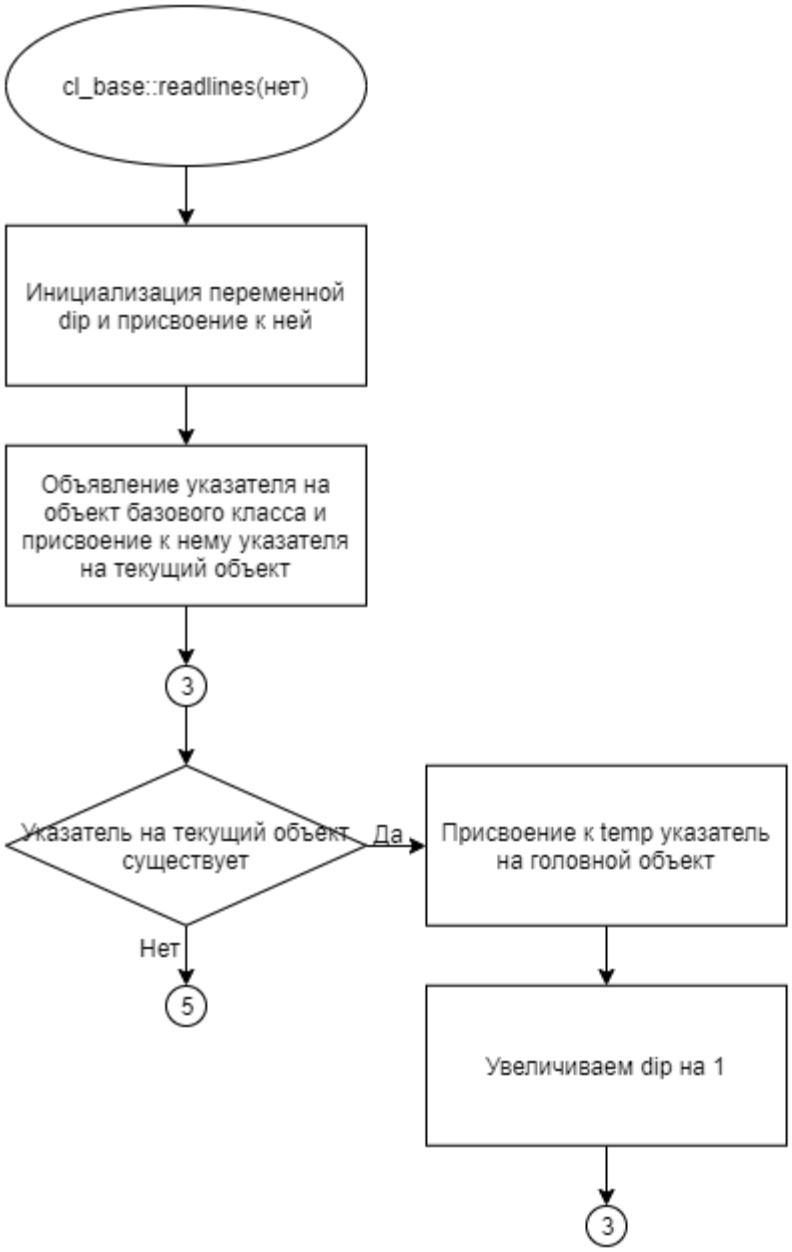

Рисунок 7 – Блок-схема алгоритма

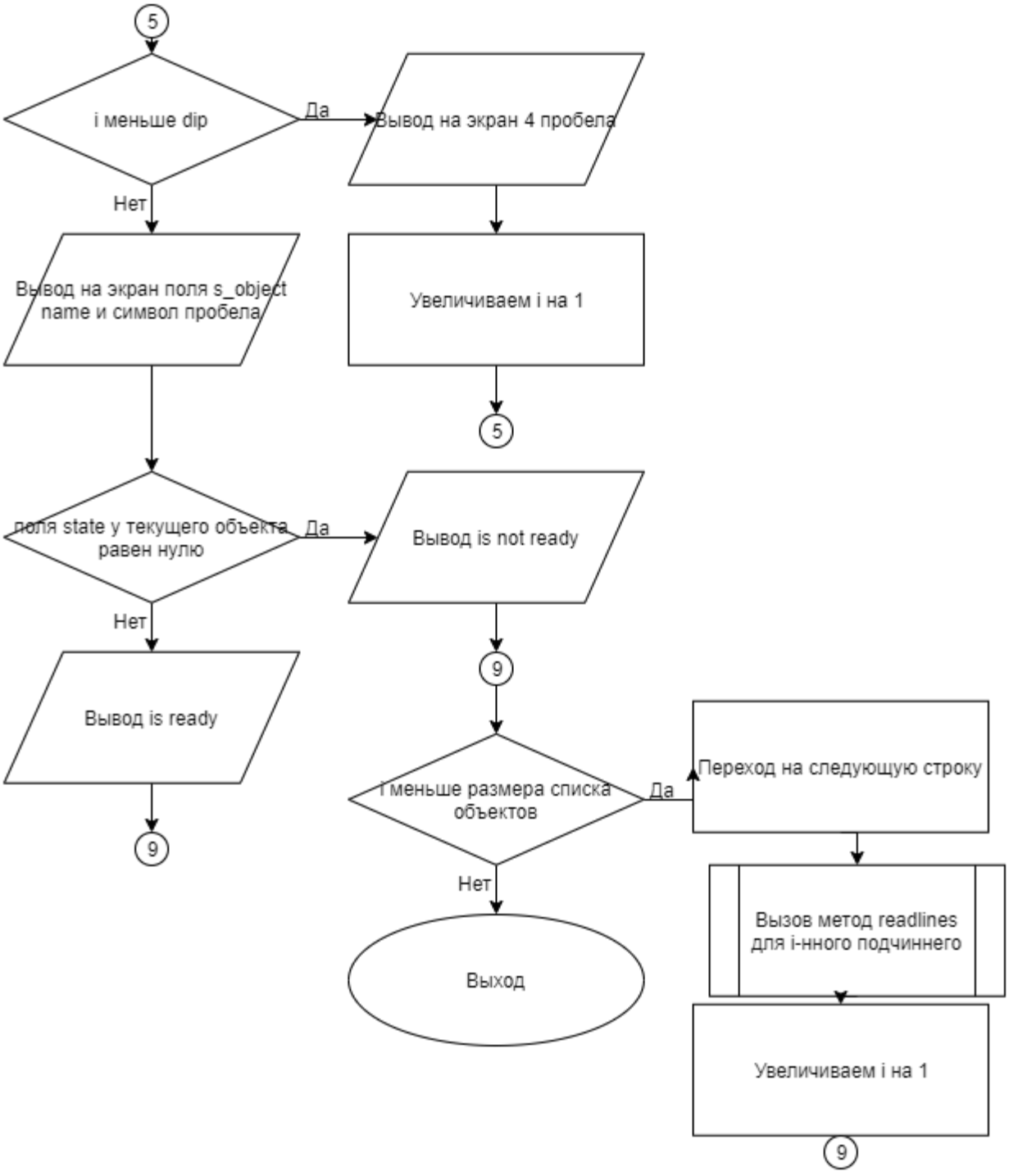

Рисунок 8 – Блок-схема алгоритма

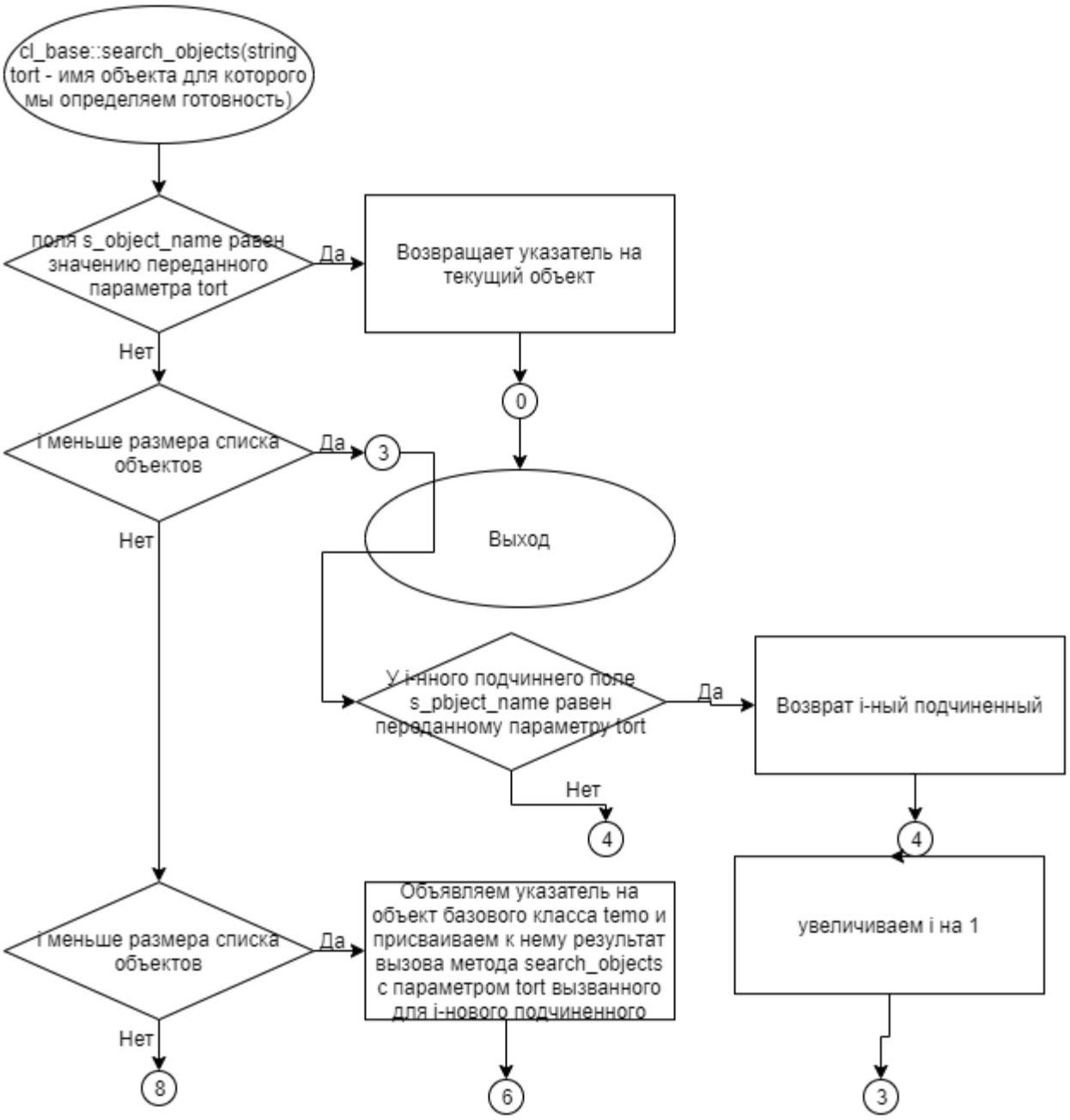

Рисунок 9 – Блок-схема алгоритма

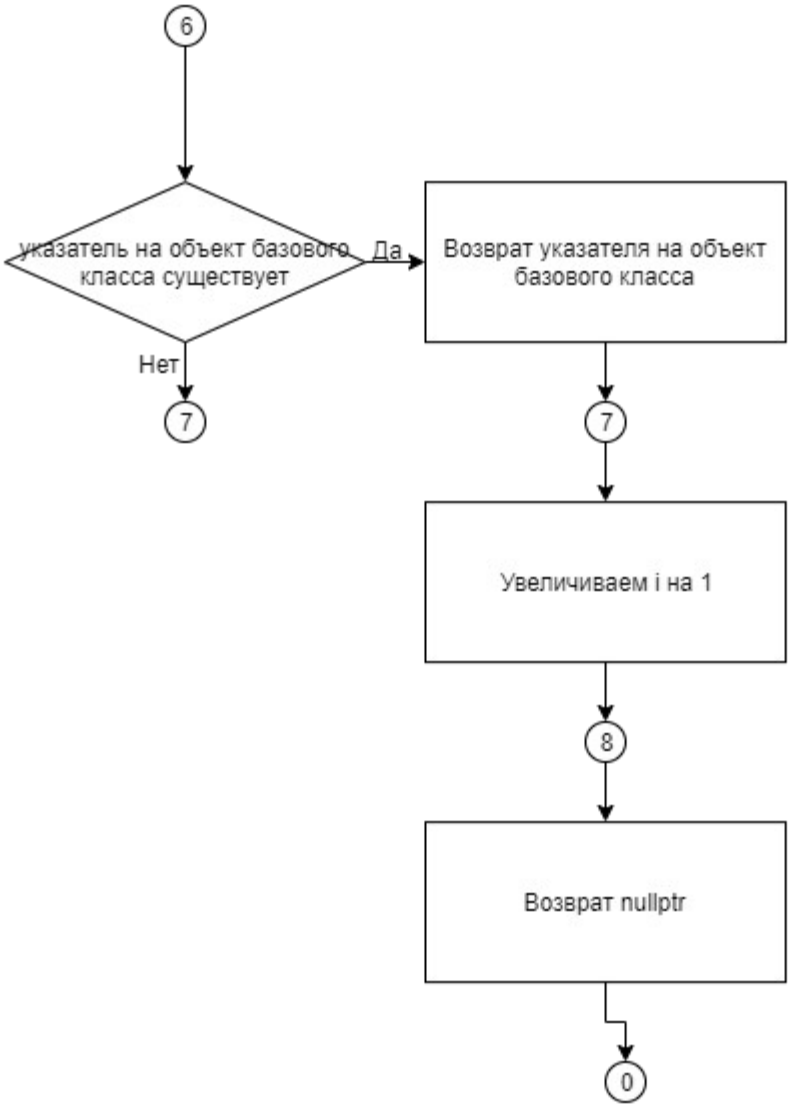

Рисунок 10 – Блок-схема алгоритма

Рисунок 11 – Блок-схема алгоритма

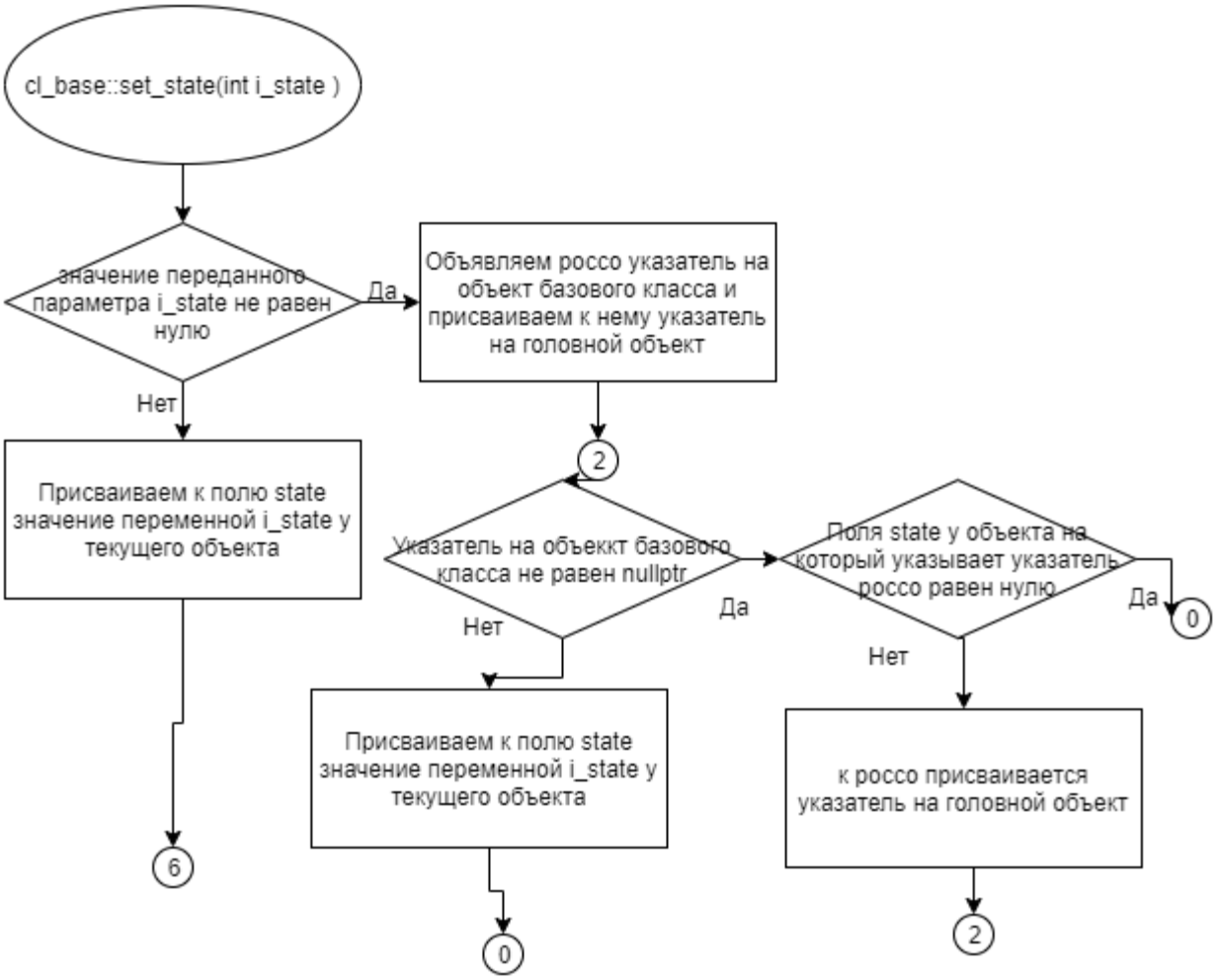

Рисунок 12 – Блок-схема алгоритма

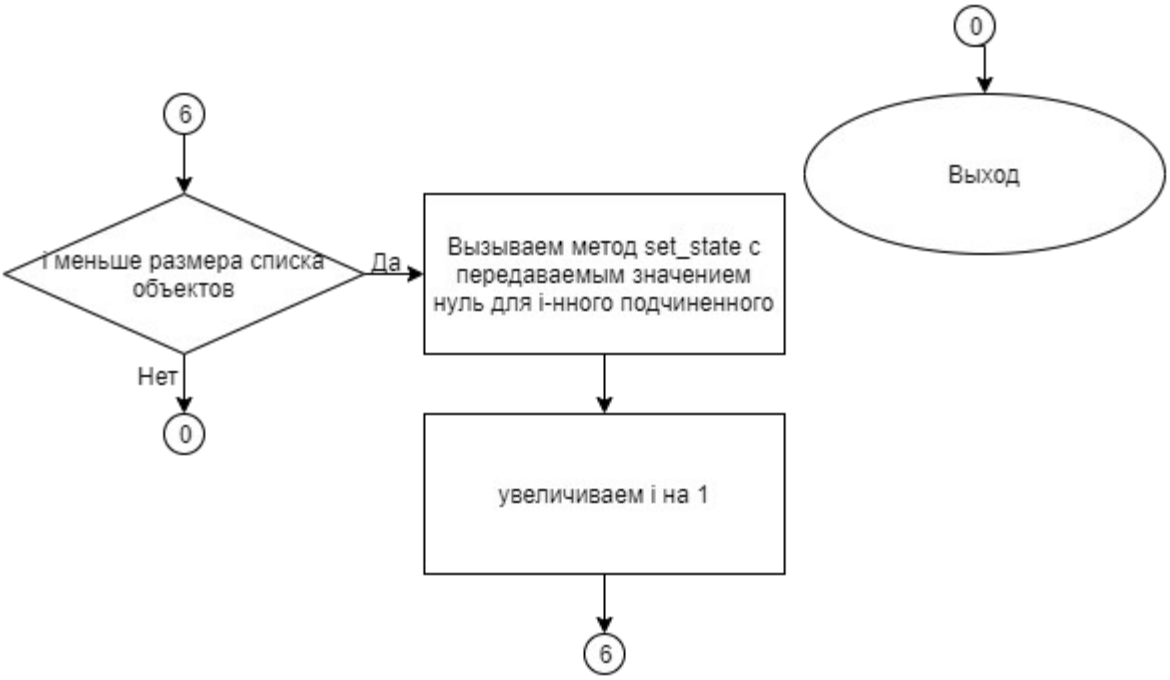

Рисунок 13 – Блок-схема алгоритма

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()
{
      cout << "Object tree\n";//<< endl;
      new_print();
      cout<<"The tree of objects and their readiness"<< endl;
      readlines();
      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);
	}
}
```

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:

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();
};
#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;
}

void cl_base::new_print()
{
      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 << endl;
      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();
      }
}
```

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>
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);
	void new_print();
	void readlines();
};

#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();  // 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 9.

Таблица 9 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
	Object tree
	app_root
	    object_01
	        object_07
	    object_02
	        object_04
	        object_05
	The tree of objects and their readiness
	app_root is ready
	    object_01 is ready
	          object_07 is not ready
	    object_02 is ready
	        object_04 is ready
	          object_05 is not ready
		Object tree
		app_root
		    object_01
		        object_07
		    object_02
		        object_04
		        object_05
		The tree of objects and their readiness
		app_root is ready
		    object_01 is ready
		          object_07 is not ready
		    object_02 is ready
		        object_04 is ready
		          object_05 is not ready

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).