Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

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

Расширить функциональность базового класса:

- Метод переопределения головного объекта для текущего в дереве иерархии. Метод должен иметь один параметр, указатель на объект базового класса, содержащий указатель на новый головной объект. Переопределение головного объект для корневого объекта недопустимо. Недопустимо создать второй корневой объект. Недопустимо при переопределении, чтобы у нового головного появились два подчиненных объекта с одинаковым наименованием. Новый головной объект не должен принадлежать к объектам из ветки текущего. Если переопределение выполнено, метод возвращает значение «истина», иначе «ложь»;
- Метод удаления подчиненного объекта по наименованию. Если объект не найден, то метод завершает работу. Один параметр строкового типа, содержит наименование удаляемого подчиненного объекта;
- Метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задаться в следующем виде:
    - / - корневой объект;
    - //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
    - . - текущий объект;
    - .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- «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_2/ob_3
/ob_1/ob_2/ob_3
```

Если координата - пустая строка или объект не найден или определяется неоднозначно (дубляж имен на ветке, на дереве), тогда вернуть нулевой указатель.

Наименование объекта не содержит символы «.» и «/».

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 необходимо соблюдать. Если это требование исходя из входных данных нарушается, то соответствующий подчиненный объект не создается.

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). Если номер класса объекта задан некорректно, то объект не создается.

Собранная система отрабатывает следующие команды:

- SET «координата» – устанавливает текущий объект;
- FIND «координата» – находит объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» задает новый головной объект;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим. При вводе данных в названии команд ошибок нет. Если при переопределении головного объекта нарушается уникальность наименований подчиненных объектов для нового головного, переопределение не производится.

1.1 Описание входных данных

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.

После ввода состава дерева иерархии построчно вводятся команды:

- SET «координата» – установить текущий объект;
- FIND «координата» – найти объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» соответствует новому головному объекту;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершить функционирование системы (выполнение программы).

Команды SET, FIND, MOVE и DELETE вводятся произвольное число раз.

Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов.

```
rootela
/ 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_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева как в 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 с кодом возврата 1.

Если при построении при попытке создания объекта обнаружен дубляж, то вывести:

```
«координата головного объекта»     Dubbing the names of subordinate objects
```

Если дерево построено, то далее построчно вводятся команды.

Для команд SET если объект найден, то вывести:

```
Object is set: «имя объекта»
```

в противном случае:

```
The object was not found at the specified coordinate: «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
```

Для команд MOVE вывести:

```
New head object: «наименование нового головного объекта»
```

Если головной объект не найден, то:

```
«искомая координата объекта»     Head object is not found
```

Если переопределить головной объект не удалось, то:

```
«искомая координата объекта»     Redefining the head object failed
```

Если у нового головного объекта уже есть подчиненный с таким же именем, то вывести:

```
«искомая координата объекта»     Dubbing the names of subordinate objects
```

При попытке переподчинения головного объекта к объекту на ветке, вывести:

```
«координата нового головного объекта»      Redefining the head object failed
```

Для команды DELETE:

Если подчиненный объект удален, то вывести:

```
The object «абсолютный путь удаленного объекта» has been deleted
```

Если объект не найден, то ничего не выводить.

После команды END с новой строки вывести:

```
Current object hierarchy tree
```

Со следующей строки вывести текущую иерархию дерева.

Пример вывода иерархии дерева объектов.

```
Object tree
rootela
    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_7     Object is not found
```

```
object_4/object_7     Object name: object_7
.     Object name: object_2
.object_7     Object name: object_7
object_4/object_7     Object name: object_7
.object_7     Redefining the head object failed
Object is set: object_7
//object_1     Dubbing the names of subordinate objects
New head object: object_3
Current object hierarchy tree
rootela
    object_1
        object_7
    object_2
        object_4
        object_5
        object_3
    object_3
        object_7
```

2 МЕТОД РЕШЕНИЯ

- Класс cl_base
  - Методы
    - find_obj_obj_by_coord()
      - Функционал: переопределение головного объекта для текущего в дереве иерархии
    - deleteSubObj()
      - Функционал: удаление починнённого объекта по имени
    - changeHeadObj()
      - Функционал: получение указателя на любой объект в составе дерева иерархии объектов согласно пути
    - printFromCurrent
      - Функционал: вывод дерева с текущего объекта

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_obj_by класса cl_base

Функционал: Получение адреса объекта базового класса по его координатам.

Параметры: Нет.

Возвращаемое значение: Указатель (адрес объекта базового класса).

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода find_obj_by класса cl_base

№	Предикат	Действия	№ перехода
1	Значение указателя s_object_path = ""	Возврат нулевого указателя (nullptr)	∅
			2
2		Объявление переменной s_path_item	3
3		Инициализация указателя head_obj адресом текущего объекта	4
4	Значение параметра s_object_path = ""	Возврат адреса текущего объекта	∅
			5
5	Значение элемента с индексом 0 параметра s_object_path = ""	Удаление элемента с индексом 0 из значения переменной s_object_path	6
			7
6		Возврат значения метода search_by_name()	∅

№	Предикат	Действия	№ перехода
		текущего объекта с переданным значением параметра s_object_path	
7	Значение элемента с индексом 1 параметра s_object_path = "/" и значение элемента с индексом 0 параметра s_object_path = "/"	Удаление элемента с индексом 0 из значения переменной s_object_path	8
			11
8		Удаление элемента с индексом 0 из значения переменной s_object_path	9
9	Значение поля p_head_object объекта, доступного через указатель head_obj != нулевому указателю	Присвоение указателю head_obj значения поля p_head_object объекта, доступного через указатель head_obj	9
			10
10		Возврат возвращаемого значения метода findObjOnBranch() объекта, доступного через указатель head_obj, с переданным значением параметра s_object_path	∅
11	Значение элемента с индексом 0 параметра s_object_path = "/"	Удаление элемента с индексом 0 из значения переменной s_object_path	12
			13
12	Значение элемента с индексом объекта, доступного через указатель head_obj != нулевому указателю	Присвоение указателю head_obj значения поля p_head_object объекта, доступного через указатель head_obj	12
			13

№	Предикат	Действия	№ перехода
13		Создание объекта стандартного потока stringstream с передачей значения параметра s_object_path	14
14	Возвращаемое значение метода getline() объекта ss_path != концу строки	Присвоение переменной s_path_item возвращаемого значения метода getline() объекта ss_path	15
			17
15		Присвоение указателю head_obj возвращаемого значения метода get_sub_obj() объекта, доступного через указатель head_obj, с переданным значением параметра s_path_item	16
16	Значение указателя head_obj != нулевому указателю	Возврат нулевого указателя	∅
			14
17		Возврат значения указателя head_obj	∅

3.2 Алгоритм метода changeHeadObj класса cl_base

Функционал: Замена головного объекта.

Параметры: Указатель на объект класса cl_base.

Возвращаемое значение: Bool (Логический нуль или единица).

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода changeHeadObj класса cl_base

№	Предикат	Действия	№ перехода
1	Значение параметра new_head_obj != нулевому указателю	2	∅

№	Предикат	Действия	№ перехода
		Возврат false	2
2	Значение указателя temp != нулевому указателю	Присвоение указателю temp значения поля p_head_object объекта, доступного через указатель temp	3
			4
3	Значение указателя temp == адресу текущего объекта	Возврат false	∅
			2
4	Возвращаемое значение метода get_sub_obj() объекта, доступного через указатель new_head_obj	Удаление элемента, равного адресу текущего объекта из массива p_sub_objects объекта, доступного через значение поля p_head_object текущего объекта	5
		Возврат false	∅
5		Добавление элемента, равного адресу текущего объекта в массиве p_sub_objects объекта, доступного через указатель new_head_obj	6
6		Присвоение полю p_head_object текущего объекта значения параметра new_head_obj	7
7		Возврат true	∅

3.3 Алгоритм метода deleteSubordinateObj класса cl_base

Функционал: Удаление подчинённого объекта текущего объекта по имени.

Параметры: Строковый name, имя объекта базового класса.

Возвращаемое значение: Void (пустое множество).

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода deleteSubordinateObj класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация указателя subordinate_obj возвращаемым значением метода hetPtrSubObj() текущего объекта с передачей параметра name	2
2	Значение указателя subordinte_obj != нулевому указателю	Удаление элемента, равного значению указателя subordinate_obj из массива subordinate_objects текущего объекта	3
			∅
3		Удаление объекта, используя оператор delete, досупного через указатель	∅

3.4 Алгоритм метода print_tree класса cl_base

Функционал: Вывод дерева с текущего объекта.

Параметры: Целочисленный параметр n.

Возвращаемое значение: Пустое множество (void).

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод имени на экран	2
2	Перебор p_sub_obj с помощью цикла	Рекурсивный вызов функции	∅
			∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

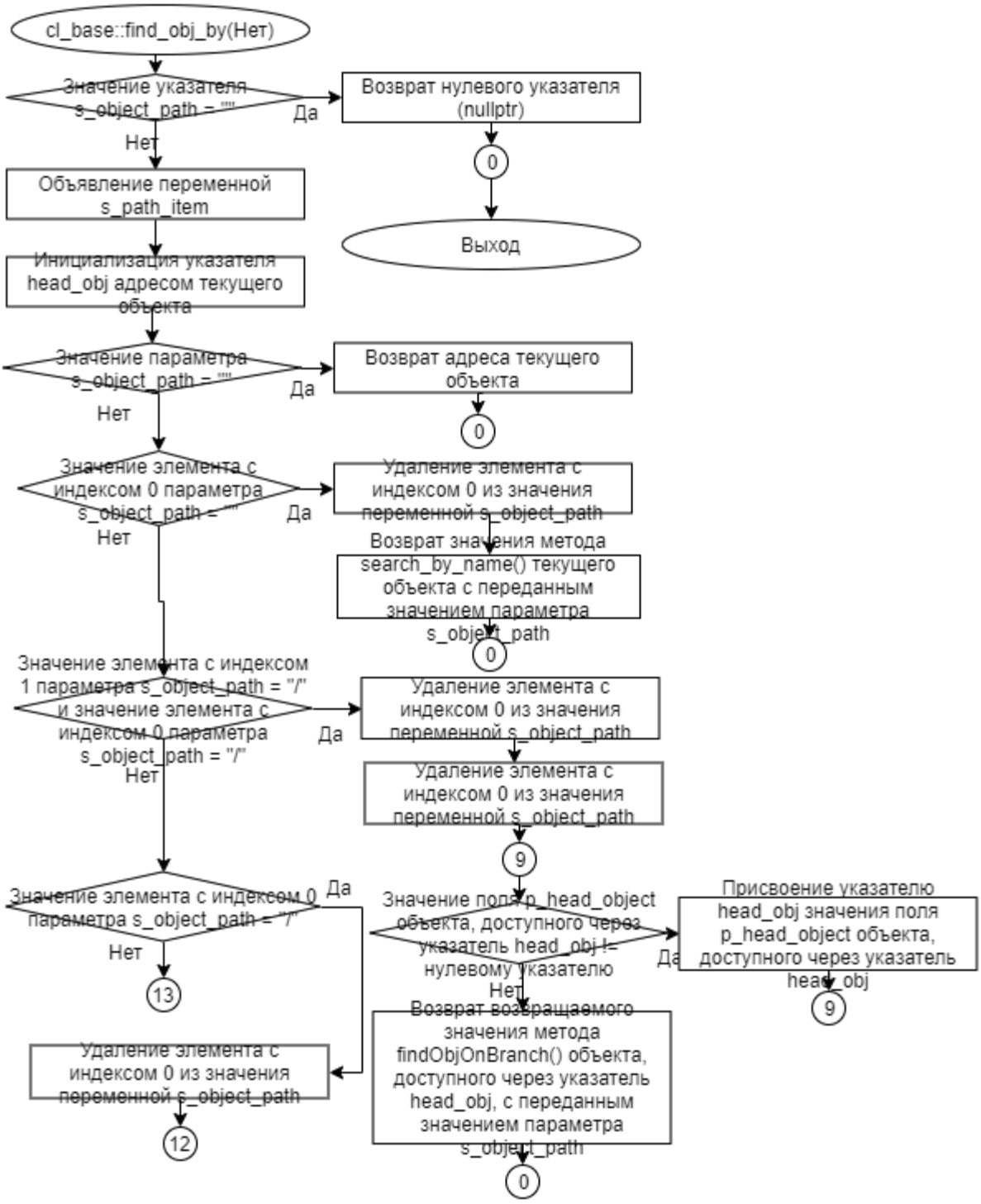

Рисунок 1 – Блок-схема алгоритма

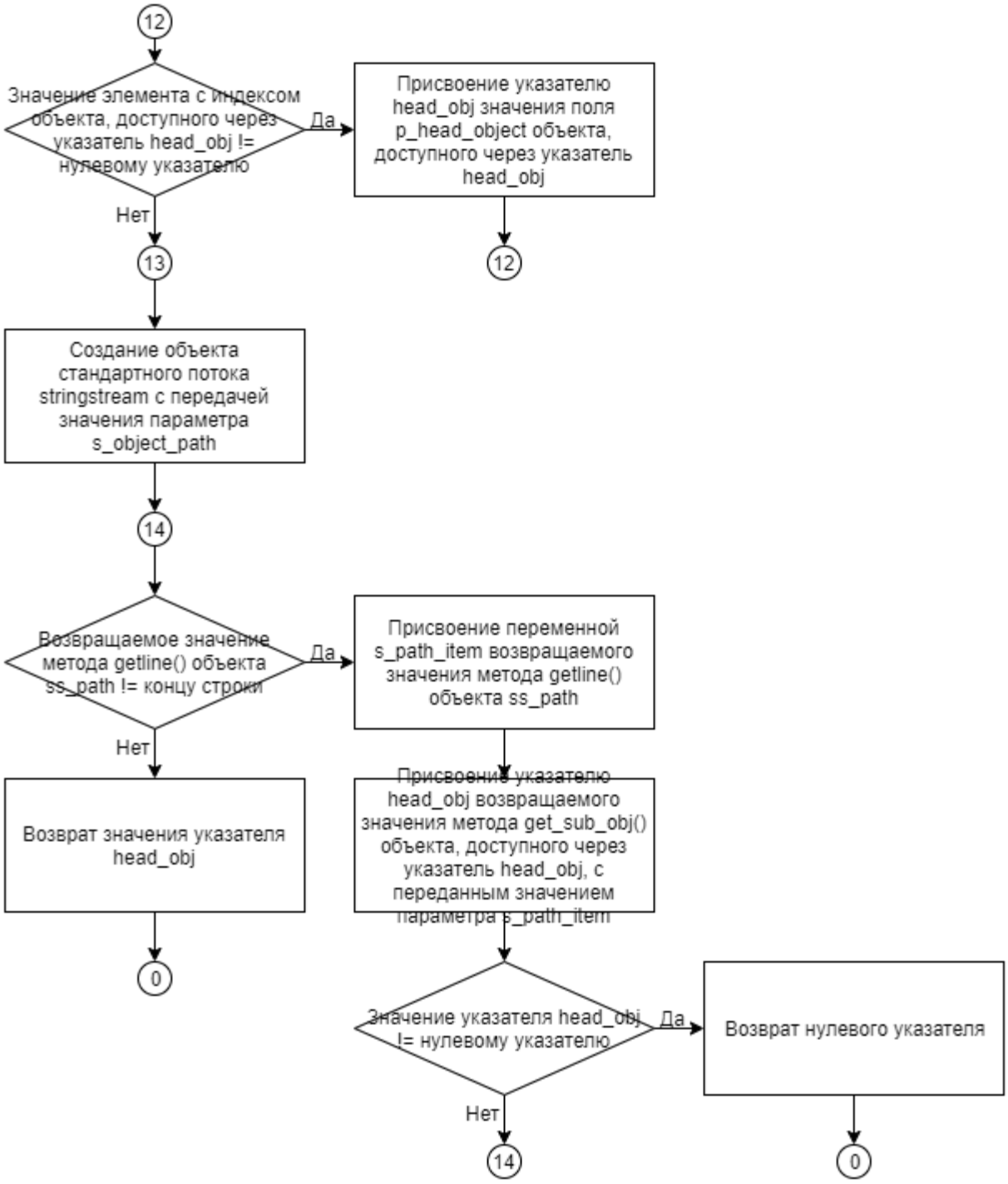

Рисунок 2 – Блок-схема алгоритма

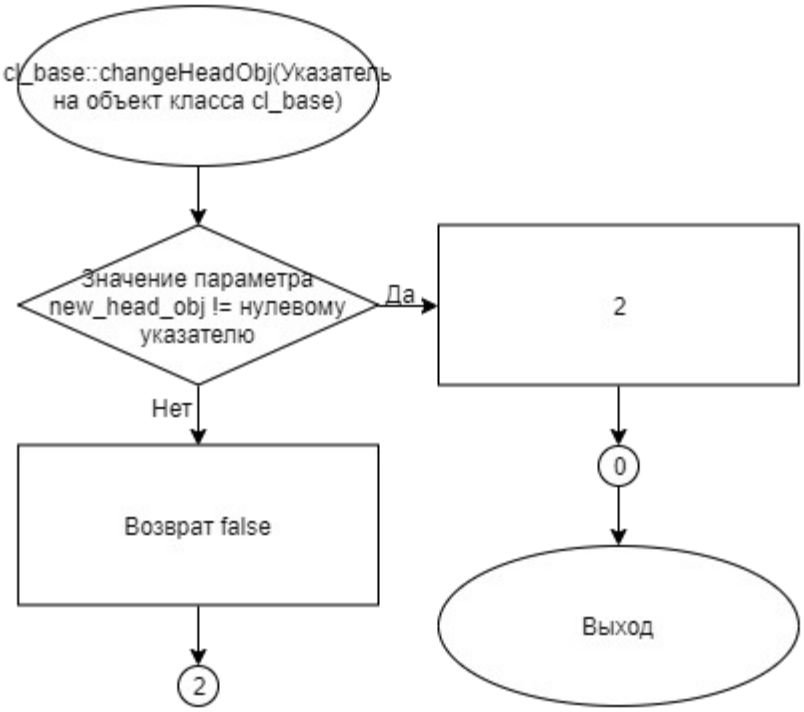

Рисунок 3 – Блок-схема алгоритма

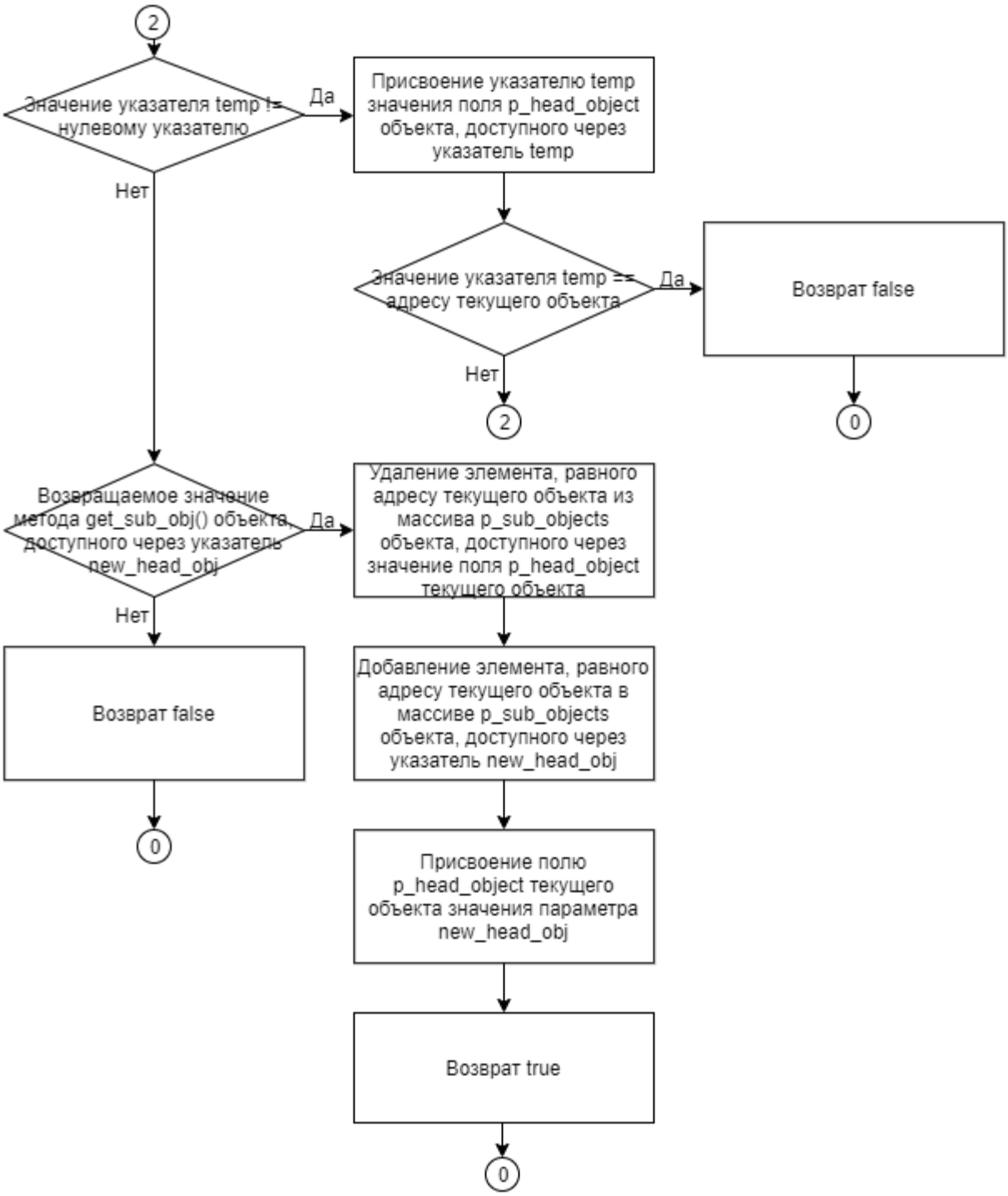

Рисунок 4 – Блок-схема алгоритма

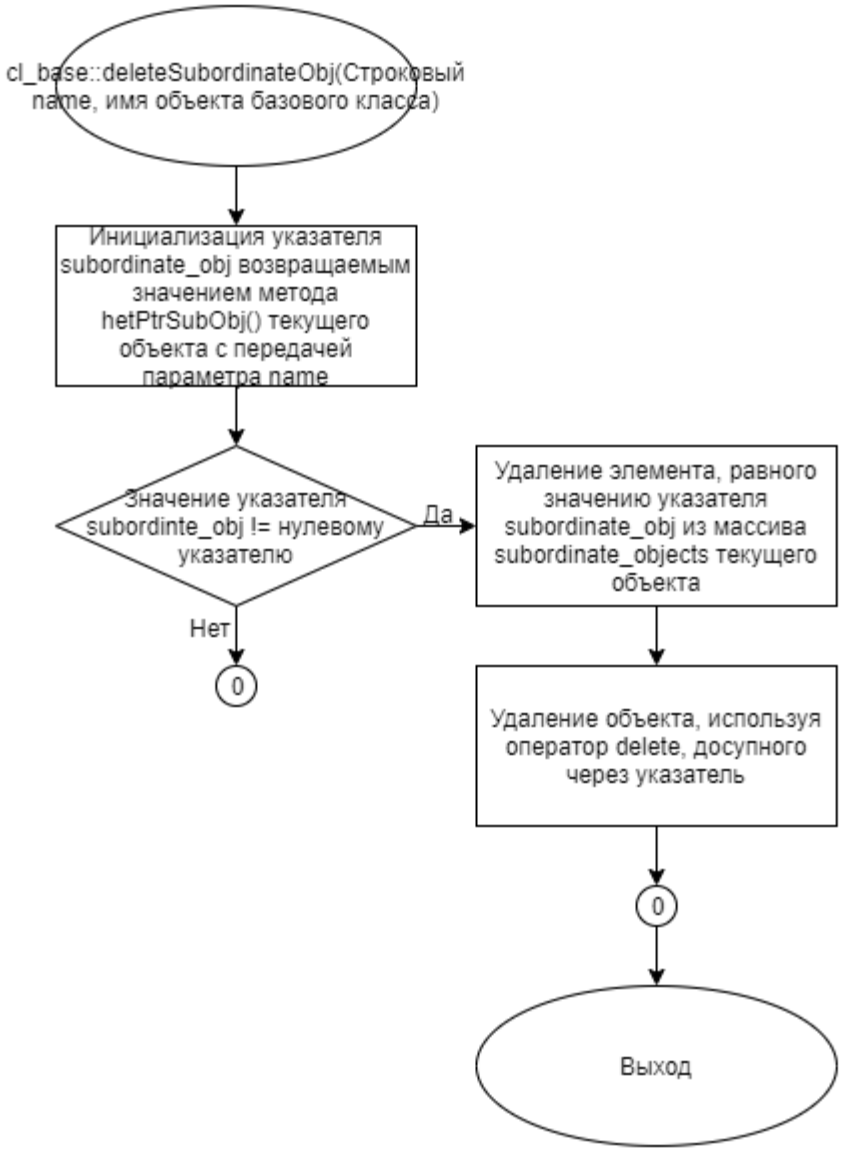

Рисунок 5 – Блок-схема алгоритма

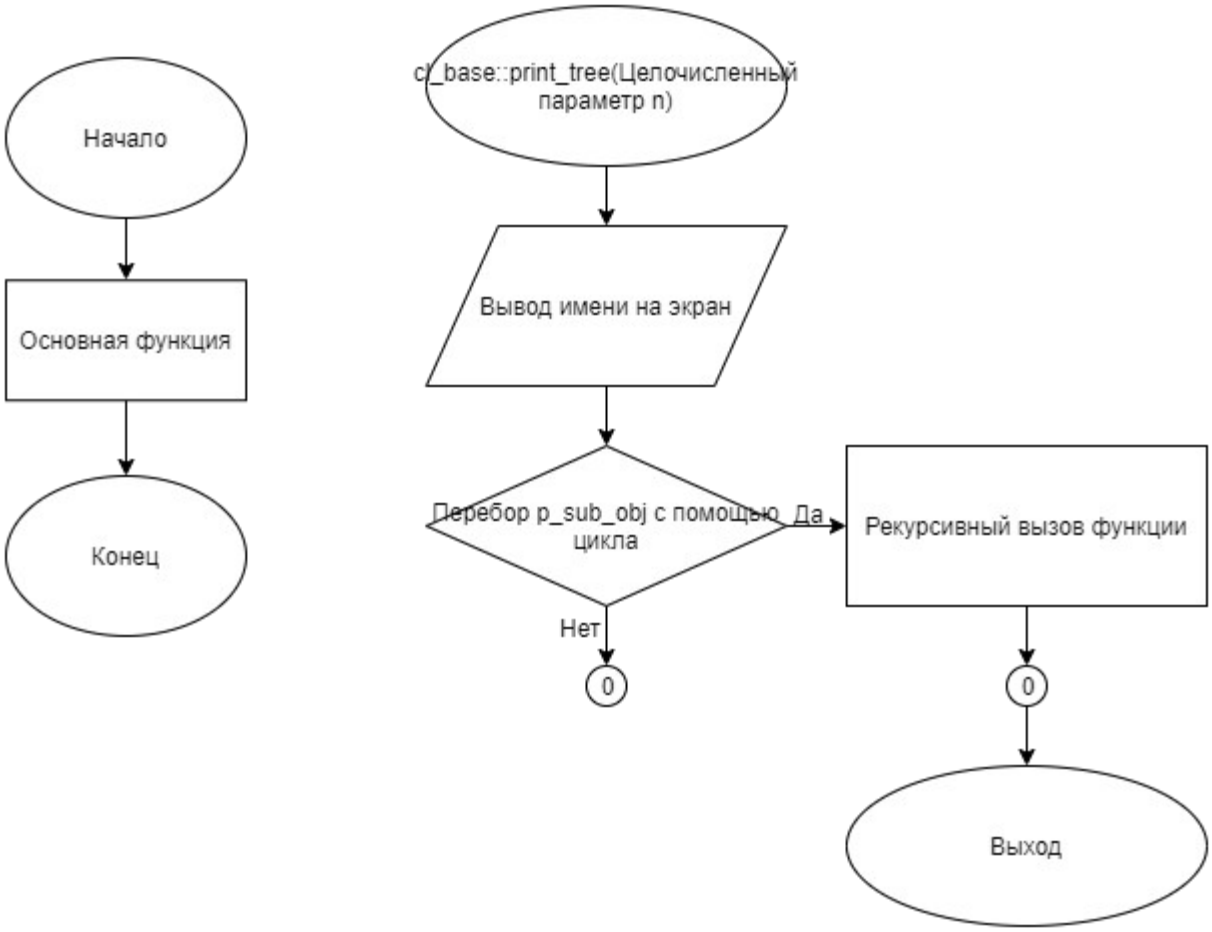

Рисунок 6 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(cl_base* p_head_object, string s_name) : cl_base(p_head_object, s_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
{
public:
      cl_1(cl_base* p_head_object, string s_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_name) : cl_base(p_head_object, s_name)
{}
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef KW_CL_2_H__
#define KW_CL_2_H__
#include "cl_base.h"
class cl_2 : public cl_base
{
public:
      cl_2(cl_base* p_head_object, string s_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_name) : cl_base(p_head_object, s_name)
{}
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef KW_CL_3_H__
#define KW_CL_3_H__
#include "cl_base.h"
class cl_3 : public cl_base
{
public:
      cl_3(cl_base* p_head_object, string s_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_name) : cl_base(p_head_object, s_name)
{}
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef KW_CL_4_H__
#define KW_CL_4_H__
#include "cl_base.h"
class cl_4 : public cl_base
{
public:
      cl_4(cl_base* p_head_object, string s_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_name) : cl_base(p_head_object, s_name)
{}
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef KW_CL_5_H__
#define KW_CL_5_H__
#include "cl_base.h"
class cl_5 : public cl_base
{
public:
      cl_5(cl_base* p_head_object, string s_name);
};
#endif
```

5.11 Файл cl_6.cpp

Листинг 11 – cl_6.cpp

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

5.12 Файл cl_6.h

Листинг 12 – cl_6.h

```
#ifndef KW_CL_6_H__
#define KW_CL_6_H__
#include "cl_base.h"
class cl_6 : public cl_base
{
public:
	cl_6(cl_base* p_head_object, string s_name);
};
#endif
```

5.13 Файл cl_application.cpp

Листинг 13 – cl_application.cpp

```
#include "cl_application.h"
cl_application::cl_application(cl_base* p_head_object) :cl_base(p_head_object) {}
void cl_application::build_tree_objects()
{
	string s_head, s_sub;
	int s_num_obj, s_redy_obj;
	cl_base* p_head = this, * p_sub = nullptr;
	cin >> s_head;
	set_name(s_head);
	while (true)
	{
		cin >> s_head;
		if (s_head == "endtree")
			break;
		cin >> s_sub >> s_num_obj;
		p_head = find_obj_by_coord(s_head);
		if (p_head != nullptr)
		{
			switch (s_num_obj)
			{
				case 1:
					p_sub = new cl_1(p_head, s_sub);
					break;
				case 2:
					p_sub = new cl_2(p_head, s_sub);
					break;
				case 3:
					p_sub = new cl_3(p_head, s_sub);
					break;
				case 4:
					p_sub = new cl_4(p_head, s_sub);
					break;
				case 5:
```

```
                              p_sub = new cl_5(p_head, s_sub);
                              break;
                        case 6:
                              p_sub = new cl_6(p_head, s_sub);
                              break;
                  }
            }
            else
            {
                  cout << "Object tree";
                  print_tree();
                  cout << endl << "The head object " << s_head << " is not found";
                  exit(1);
            }
      }
}
void cl_application::build_commands()
{
      string command;
      string object_name;
      cl_base* current_obj = this;
      while (command != "END")
      {
            cin >> command;
            cin >> object_name;
            cl_base* temp = current_obj->find_obj_by_coord(object_name);
            if (command == "SET")
            {
                  if (temp != nullptr)
                  {
                        current_obj = temp;
                        cout << "Object is set: " << current_obj->get_name() <<
endl;
                  }
                  else
                        cout << "The object was not found at specified
coordinate:" << object_name << endl;
            }
            if (command == "FIND")
            {
                  if (temp != nullptr)
                  {
                        cout << object_name << "       Object name: " << temp-
>get_name() << endl;
                  }
                  else
                  {
                        cout << object_name << "     Object is not found" << endl;
                  }
            }
            if (command == "MOVE")
            {
                  if (temp != nullptr)
                  {
                        if (current_obj->change_head_obj(temp))
                        {
                              cout << "New head object: " << temp->get_name() <<
```

```
endl;
                        }
                        else if (temp->get_sub_obj(current_obj->get_name()))
                        {
                              cout << object_name << "      Dubbing the names of
subordinate objects" << endl;
                        }
                        else
                        {
                              cout << object_name << "       Redefining the head
object failed" << endl;
                        }
                  }
                  else
                  {
                        cout << object_name << "     Head object is not found" <<
endl;
                  }
            }
            if (command == "DELETE")
            {
                  if   (current_obj->search_by_name(object_name)   !=   nullptr   &&
current_obj->get_name() != object_name)
                  {
                        vector <cl_base*> path;
                        cl_base* temp2 = current_obj;
                        while (temp2->get_head() != nullptr)
                        {
                              path.insert(path.begin(), temp2);
                              temp2 = temp2->get_head();
                        }
                        path.insert(path.end(),                         current_obj-
>search_by_name(object_name));
                        cout << "The object ";
                        for (auto path_item : path)
                        {
                              cout << "/" << path_item->get_name();
                        }
                        cout << " has been deleted" << endl;
                        current_obj->delete_subordinate_obj(object_name);
                  }
            }
      }
}
int cl_application::exec_app()
{
      cout << "Object tree";
      print_tree();
      cout << endl;
      build_commands();
      cout << "Current object hierarchy tree";
      print_tree();
      return 0;
}
```

5.14 Файл cl_application.h

Листинг 14 – cl_application.h

```
#ifndef __CL_APPLICATION_H__
#define __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"
#include "cl_6.h"
class cl_application : public cl_base
{
public:
	cl_application(cl_base* p_head_object);
	void build_tree_objects();
	int exec_app();
	void build_commands();
};
#endif
```

5.15 Файл cl_base.cpp

Листинг 15 – cl_base.cpp

```
#include "cl_base.h"
cl_base::cl_base(cl_base* p_head_object, string s_name)
{
	this->s_name = s_name;
	this->p_head_object = p_head_object;
	if (p_head_object != nullptr)
		p_head_object->p_sub_objects.push_back(this);
}
cl_base::~cl_base()
{
	for (int i = 0; i < p_sub_objects.size(); i++)
		delete p_sub_objects[i];
}
bool cl_base::set_name(string s_new_name)
{
	if (get_head() != nullptr)
	{
		for (int i = 0; i < get_head()->p_sub_objects.size(); i++)
		{
			if (get_head()->p_sub_objects[i]->get_name() == s_new_name)
			{
				return false;
			}
		}
```

```
	}
	s_name = s_new_name;
	return true;
}
void cl_base::print_current(string delay)
{
	cout << endl << delay;
	get_ready(get_name());
	for (auto p_sub : p_sub_objects)
		p_sub->print_current(delay + "    ");
}
string cl_base::get_name()
{
	return s_name;
}
cl_base* cl_base::get_head()
{
	return p_head_object;
}
cl_base* cl_base::get_sub_obj(string s_name)
{
	for (int i = 0; i < p_sub_objects.size(); i++)
	{
		if (p_sub_objects[i]->s_name == s_name)
		{
			return p_sub_objects[i];
		}
	}
	return nullptr;
}
int cl_base::count(string name)
{
	int count = 0;
	if (get_name() == name)
		count++;
	for (int i = 0; i < p_sub_objects.size(); i++)
		count += p_sub_objects[i]->count(name);
	return count;
}
cl_base* cl_base::search_by_name(string name)
{
	if (s_name == name)
		return this;
	cl_base* p_result = nullptr;
	for (int i = 0; i < p_sub_objects.size(); i++)
	{
		p_result = p_sub_objects[i]->search_by_name(name);
		if (p_result != nullptr)
			return p_result;
	}
	return nullptr;
}
cl_base* cl_base::search_cur(string name)
{
	if (count(name) != 1)
		return nullptr;
	return search_by_name(name);
```

```
}
cl_base* cl_base::search_from_root(string name)
{
	if (p_head_object != nullptr)
		return p_head_object->search_from_root(name);
	else
		return search_cur(name);
}
void cl_base::set_ready(int s_new_ready)
{
	if (s_new_ready != 0)
	{
		if (p_head_object == nullptr || p_head_object != nullptr &&
p_head_object->p_ready != 0)
		{
		p_ready = s_new_ready;
		}
	}
	else
	{
		p_ready = s_new_ready;
		for (int i = 0; i < p_sub_objects.size(); i++)
			p_sub_objects[i]->set_ready(s_new_ready);
	}
}
void cl_base::get_ready(string name)
{
	if (get_name() == name)
	{
		if (p_ready != 0)
		{
			cout << get_name() << " is ready";
		}
		else
		{
			cout << get_name() << " is not ready";
		}
	}
	else
	{
		for (int i = 0; i < p_sub_objects.size(); i++)
		{
			return p_sub_objects[i]->get_ready(name);
		}
	}
}
bool cl_base::change_head_obj(cl_base* new_head_obj)
{
	if (new_head_obj != nullptr)
	{
		cl_base* temp = new_head_obj;
		while (temp != nullptr)
		{
			temp = temp->p_head_object;
			if (temp == this)
			{
				return false;
```

```
                  }
            }
            if     (new_head_obj->get_sub_obj(get_name())     ==     nullptr     &&
p_head_object != nullptr)
            {
                  p_head_object->p_sub_objects.erase(find(p_head_object-
>p_sub_objects.begin(), p_head_object->p_sub_objects.end(), this));
                  new_head_obj->p_sub_objects.push_back(this);
                  p_head_object = new_head_obj;
                  return true;
            }
      }
      return false;
}
void cl_base::delete_subordinate_obj(string name)
{
      cl_base* subordinate_obj = get_sub_obj(name);
      if (subordinate_obj != nullptr)
      {
            p_sub_objects.erase(find(p_sub_objects.begin(),    p_sub_objects.end(),
subordinate_obj));
            delete subordinate_obj;
      }
}
cl_base* cl_base::find_obj_by_coord(string s_object_path)
{
      if (s_object_path.size() == 0)
      {
            return nullptr;
      }
      cl_base* head_obj = this;
      string s_path_item;
      if (s_object_path == "." || s_object_path == "/")
      {
            return head_obj;
      }
      if (s_object_path[0] == '.')
      {
            s_object_path.erase(s_object_path.begin());
            return search_by_name(s_object_path);
      }
      if (s_object_path[1] == '/' && s_object_path[0] == '/')
      {
            s_object_path.erase(s_object_path.begin());
            s_object_path.erase(s_object_path.begin());
            return this->search_from_root(s_object_path);
      }
      if (s_object_path[0] == '/')
      {
            s_object_path.erase(s_object_path.begin());
            while (head_obj->p_head_object != nullptr)
            {
                  head_obj = head_obj->p_head_object;
            }
      }
      stringstream ss_path(s_object_path);
      while (getline(ss_path, s_path_item, '/'))
```

```
	{
		head_obj = head_obj->get_sub_obj(s_path_item);
		if (head_obj == nullptr)
		{
			return nullptr;
		}
	}
	return head_obj;
}
void cl_base::print_tree(int n)
{
	cout << endl;
	for (int i = 0; i < n; i++)
	{
		cout << "    ";
	}
	cout << s_name;
	for (auto p_subordinate_object : p_sub_objects)
	{
		p_subordinate_object->print_tree(n + 1);
	}
}
```

5.16 Файл cl_base.h

Листинг 16 – cl_base.h

```
#ifndef __CL_BASE_H__
#define __CL_BASE_H__
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <algorithm>
using namespace std;
class cl_base
{
private:
	string s_name;
	cl_base* p_head_object;
	vector <cl_base*> p_sub_objects;
	int p_ready = 0;
public:
	cl_base(cl_base* p_head_object, string s_name = "Base Object");
	bool set_name(string s_new_name);
	string get_name();
	cl_base* get_head();
	cl_base* get_sub_obj(string s_name);
	~cl_base();
	int count(string name);
	cl_base* search_by_name(string name);
	cl_base* search_cur(string name);
	cl_base* search_from_root(string name);
```

```
      void set_ready(int s_new_ready);
      void get_ready(string name);
      void print_current(string delay = "");
      bool change_head_obj(cl_base* new_head_obj);
      void delete_subordinate_obj(string name);
      cl_base* find_obj_by_coord(string s_object_path);
      void print_tree(int n = 0);
};
#endif
```

5.17 Файл main.cpp

Листинг 17 – main.cpp

```
#include "cl_application.h"
int main()
{
      cl_application ob_cl_application(nullptr);
      ob_cl_application.build_tree_objects();
      return ob_cl_application.exec_app();
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 5.

Таблица 5 – Результат тестирования программы

Входные данные	**Ожидаемые выходные данные**	**Фактические выходные данные**

rootela
/ object_1 3
/ object_2 2
endtree
FIND //object_1
END
	Object tree
	rootela
	    object_1
	    object_2
	//object_1 Object name: object_1
	Current object hierarchy tree
	rootela
	    object_1
	    object_2
		Object tree
		rootela
		    object_1
		    object_2
		//object_1 Object name: object_1
		Current object hierarchy tree
		rootela
		    object_1
		    object_2

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).