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:

- Функционал:
  - Метод set_parent - метод переопределения головного объекта для текущего в дереве иерархии;
  - Метод remove_child_by_name - метод удаления подчиненного объекта по наименованию;
  - Метод get_object_by_path - метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты).

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 функции main

Функционал: Основная программа.

Параметры: .

Возвращаемое значение: int - код возврата.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта ob_application класса application с использованием параметрезированного конструктора и передачей в него в качестве параметра пустого указателя	2
2		Вызов метода build_tree_objects объекта ob_application	3
3		Возвращение результата работы метода exec_app() для объекта ob_application	∅

3.2 Алгоритм метода set_parent класса cl_base

Функционал: переопределение головного объекта для текущего в дереве иерархии.

Параметры: cl_base* new_parent.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода set_parent класса cl_base

№	Предикат	Действия	№ перехода
1	Головной объект совпадает с новым головным	Возвратить true	∅
			2
2	Объект является корнем или new_parent - нулевой указатель	Возвратить false	∅
			3
3	У нового головного объекта уже есть подчиненный с именем текущего объекта	Возвратить false	∅
			4
4		Объявление стека st указателей на объект класса cl_base	5
5		Добавление в стек текущего объекта	6
6	Стек содержит элементы		7
			13
7		Инициализация указателя current_node_ptr на объект класса cl_base значением верхнего элемента стека st	8
8		Удаление верхнего элемента стека st	9
9	current_node_ptr == new_parent	Возвратить ложь	∅
			10
10		Инициализация целочисленной переменной i значением 0	11
11	i < размера вектора children объекта current_node_ptr	Добавление в стек элемент children[i] объекта current_node_ptr	12

№	Предикат	Действия	№ перехода
			6
12		i += 1	11
13		Инициализация ссылки на вектор v, содержащего указатели на объекты класса cl_base значением свойства children головного объекта	14
14		Инициализация целочисленной переменной i значением 0	15
15	i < размера вектора v		16
		Возвратить false	∅
16	Имя i-го объекта равно имени текущего объекта	Удаление i-го элемента вектора v	18
			17
17		i += 1	15
18		Добавить текущий объект в вектор подчинённых объектов нового родительского объекта	19
19		Возвратить true	∅

3.3 Алгоритм метода remove_child_by_name класса cl_base

Функционал: удаление подчиненного объекта по наименованию.

Параметры: string child_name.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода remove_child_by_name класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация ссылки на вектор v, содержащий указатели на объекты класса cl_node значением свойства children	2
2		Инициализация целочисленной переменной i значением 0	3
3	i меньше размера вектора v		4
			∅
4	Имя i-го объекта равно имени текущего объекта	Вызов деструктора для объекта по i-му указателю вектора v	6
			5
5		i += 1	6
6		Удаление i-го элемента вектора v	∅

3.4 Алгоритм метода get_object_by_path класса cl_base

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

Параметры: string path.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода get_object_by_path класса cl_base

№	Предикат	Действия	№ перехода
1	path - пустая строка	Возврат nullptr	∅
			2
2	path == "."	Вернуть текущий объект	∅
			3
3	Первый символ path равен '.'	Вернуть результат вызова метода findObjOnBranch	∅

№	Предикат	Действия	№ перехода
		с параметром path, начиная со второго символа	
			4
4	Первые два символа path равны "//"	Вернуть результат вызова метода findObjOnTree с параметром path, начиная с третьего символа	∅
			5
5	Первый символ path не равен '/'	Инициализация знаковой целочисленной переменной slash_index позицией первого символа '/' в строке path	6
			9
6		Инициализация указателя child_ptr на объект класса cl_base результатом вызова метода get_child_by_name с параметром path, до первого символа '/'.	7
7	child_ptr == nullptr или в строке path нет символа '/'	Возврат child_ptr	∅
		Вернуть результат вызова метода get_object_by_path объекта child_ptr с параметром path, начиная после символа '/'	8
8		Вернуть результат вызова метода get_object_by_path объекта child_ptr с параметром path, начиная после символа '/'	∅
9		Инициализация указателя root_ptr адресом текущего объекта	10
10	у объекта по указателю root_ptr есть родитель	Присваивание root_ptr значение головного объекта по указателю root_ptr	10
			11
11	path == "/"	Возврат root_ptr	∅
			12
12		Вернуть результат вызова метода get_object_by_path объекта по указателю root_ptr с	∅

№	Предикат	Действия	№ перехода
		параметром path, начиная со второго символа	

3.5 Алгоритм метода exec_app класса application

Функционал: Запуск системы.

Параметры: .

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		Объявление строк command и input	2
2		Инициализация указателя current_object_ptr на объект класса cl_base адресом текущего объекта	3
3		Объявление указателя extra_object_ptr на объект класса cl_base	4
4		Объявление стека строк st	5
5		Вывод "Object tree"	6
6		Вызов метода printBranch	7
7		Ввод значение переменной command	8
8	command не равно "END"	Ввод input	9
			24
9	command равно "SET"		10
	command равно "FIND"		13
	command равно "MOVE"		15
	command равно "DELETE"		17
			7
10		Присваивание extra_object_ptr результат вызова метода get_object_by_path объекта по указателю	11

№	Предикат	Действия	№ перехода
		current_object_ptr с параметром path	
11	extra_object_ptr ненулевой указатель	current_object_ptr = extra_object_ptr	12
		Вывод "The object was not found at the specified coordinate: {input}" с новой строки	7
12		Вывод "Object is set: {имя объекта по указателю current_object_ptr}" с новой строки	7
13		Присваивание extra_object_ptr результат вызова метода get_object_by_path объекта по указателю current_object_ptr с параметром path	14
14	extra_object_ptr ненулевой указатель	Вывод "{input} Object name: {имя объекта по указателю extra_object_ptr}" с новой строки	7
		Вывод "{input} Object is not found" с новой строки	7
15		Присваивание extra_object_ptr результат вызова метода get_object_by_path объекта по указателю current_object_ptr с параметром path	16
16	результат вызова метода set_parent объекта по указателю current_object_ptr с параметром extra_object_ptr - истина	Вывод "New head object: {имя объекта по указателю extra_object_ptr}" с новой строки	7
	extra_object_ptr нулевой указатель	Вывод "{input} Head object is not found" с новой строки	7
	У объекта по указателю extra_object_ptr есть подчинённый с именем объекта по указателю	Вывод "{input} Dubbing the names of subordinate objects" с новой строки	7

№	Предикат	Действия	№ перехода
	current_object_ptr		
		Вывод "{input} Redefining the head object failed" с новой строки	7
17		Присваивание extra_object_ptr результат вызова метода get_child_by_name объекта по указателю current_object_ptr с параметром input	18
18	extra_object_ptr ненулевой указатель		19
			7
19	у объекта по указателю extra_object_ptr есть головной объект	добавление имени объекта по указателю extra_object_ptr на вершину стека st	20
		Вызов метода remove_child_by_name у объекта по указателю current_object_ptr с параметром input	21
20		Присваивание extra_object_ptr адрес головного объекта extra_object_ptr	19
21		Вывод "The object " с новой строки	22
22	Стек st содержит элементы	Вывод "/{вершина стека st}"	23
		Вывод " has been deleted"	7
23		Извлечение вершины стека	22
24		Вывод "Current object hierarchy tree" с новой строки	25
25		Вызов метода printBranch	26
26		Возвратить 0	∅

3.6 Алгоритм метода build_tree_objects класса application

Функционал: Строит дерево иерархии.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода build_tree_objects класса application

№	Предикат	Действия	№ перехода
1		Объявление строк path, child_name	2
2		Объявление целочисленной переменной tmp	3
3		Ввод child_name	4
4		Вызов метода setName с параметром child_name	5
5		Объявление указателя parent_node_ptr	6
6		Инициализация указателя last_created_node_ptr указателем текущего объекта	7
7		Ввод path	8
8	path не равно "endtree"	Ввод child_name и tmp	9
			∅
9		Присваивание parent_node_ptr результат вызова метода get_object_by_path с параметром path объекта по указателю last_created_node_ptr	10
10	parent_node_ptr ненулевой указатель и у объекта по этому указателю нет подчинённых с именем child_name		11
			12
11	tmp равно 1	Создание объекта класса application с помощью	12

№	Предикат	Действия	№ перехода
1		оператора new и вызова конструктора с параметрами parent_node_ptr и child_name и присваивание указателю last_cerated_node_ptr адрес этого объекта	
	tmp равно 2	Создание объекта класса cl_2 с помощью оператора new и вызова конструктора с параметрами parent_node_ptr и child_name и присваивание указателю last_cerated_node_ptr адрес этого объекта	12
	tmp равно 3	Создание объекта класса cl_3 с помощью оператора new и вызова конструктора с параметрами parent_node_ptr и child_name и присваивание указателю last_cerated_node_ptr адрес этого объекта	12
	tmp равно 4	Создание объекта класса cl_4 с помощью оператора new и вызова конструктора с параметрами parent_node_ptr и child_name и присваивание указателю last_cerated_node_ptr адрес этого объекта	12
	tmp равно 5	Создание объекта класса cl_5 с помощью оператора new и вызова конструктора с параметрами parent_node_ptr и child_name и присваивание указателю last_cerated_node_ptr адрес этого объекта	12
	tmp равно 6	Создание объекта класса cl_6 с помощью оператора new и вызова конструктора с параметрами parent_node_ptr и child_name и присваивание указателю last_cerated_node_ptr адрес этого объекта	12

№	Предикат	Действия	№ перехода
			12
1 2		Ввод path	8

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

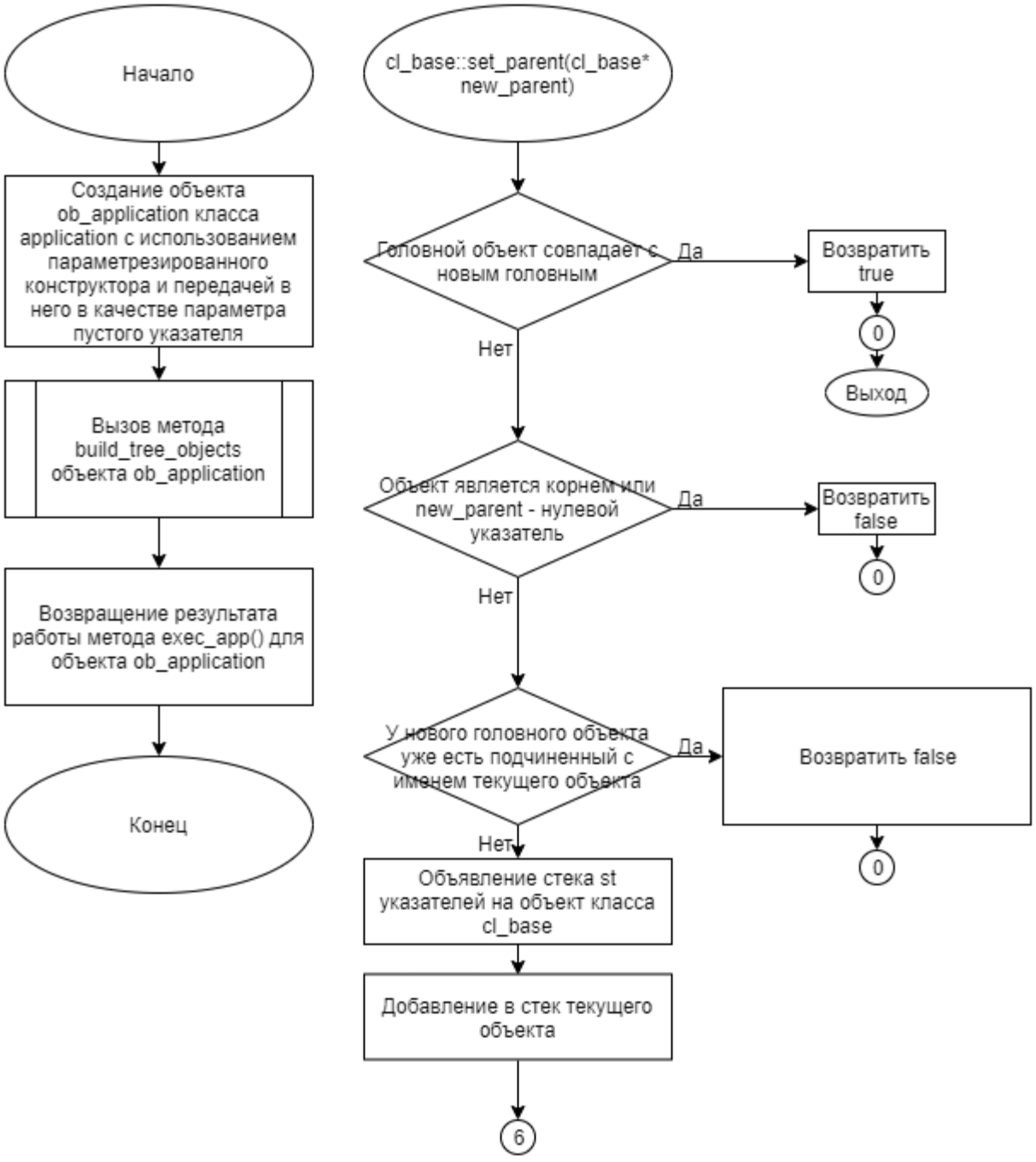

Рисунок 1 – Блок-схема алгоритма

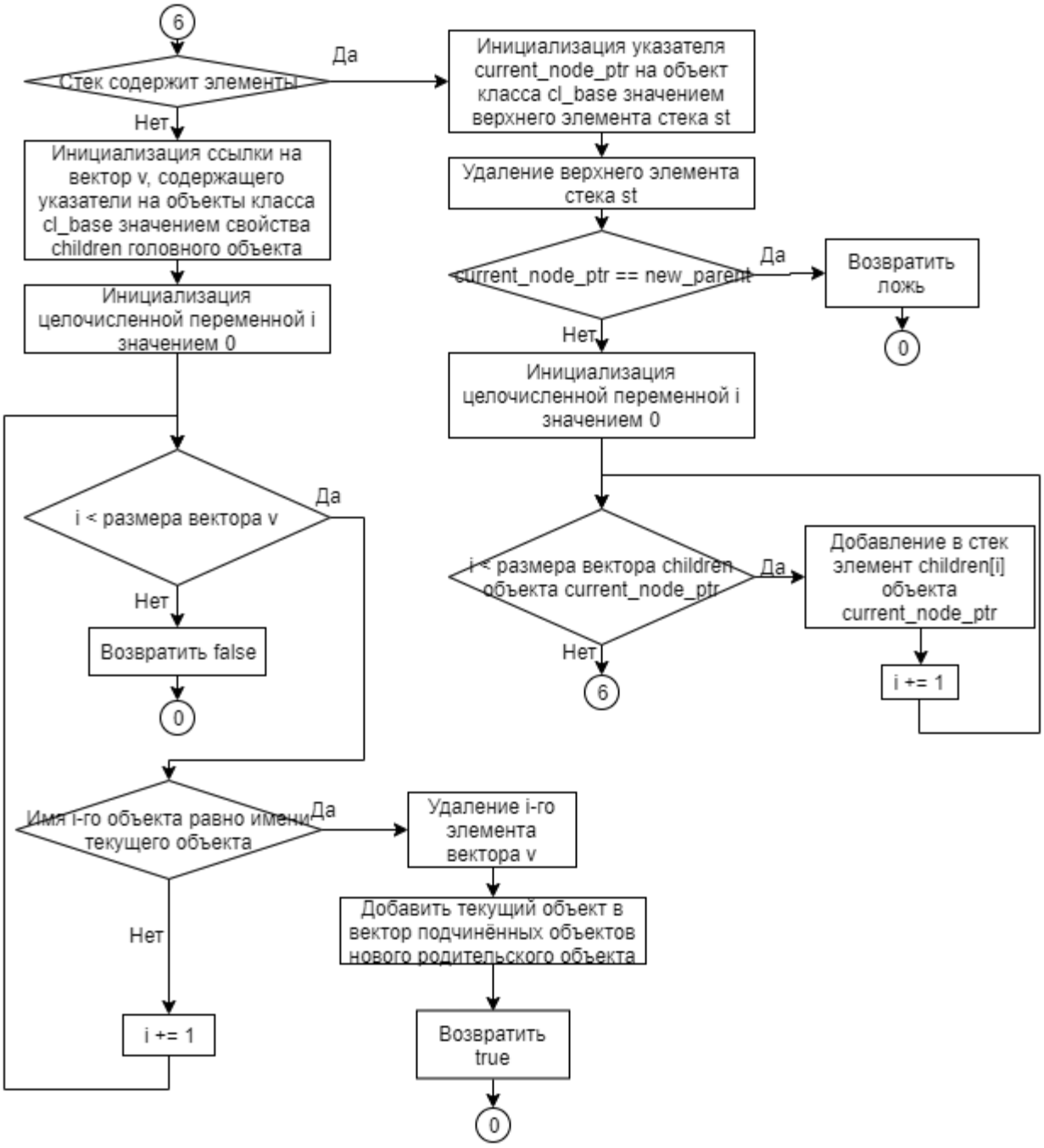

Рисунок 2 – Блок-схема алгоритма

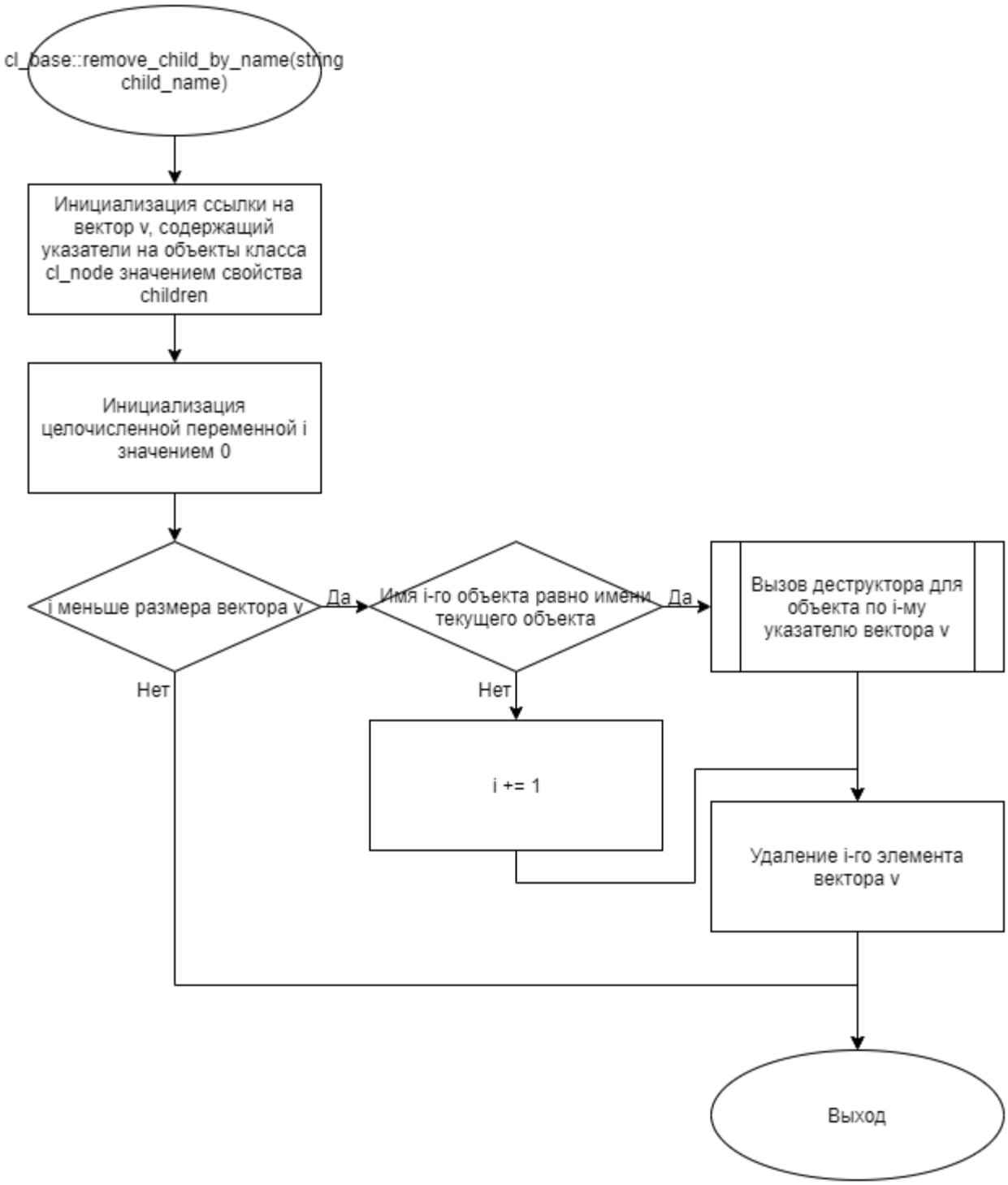

Рисунок 3 – Блок-схема алгоритма

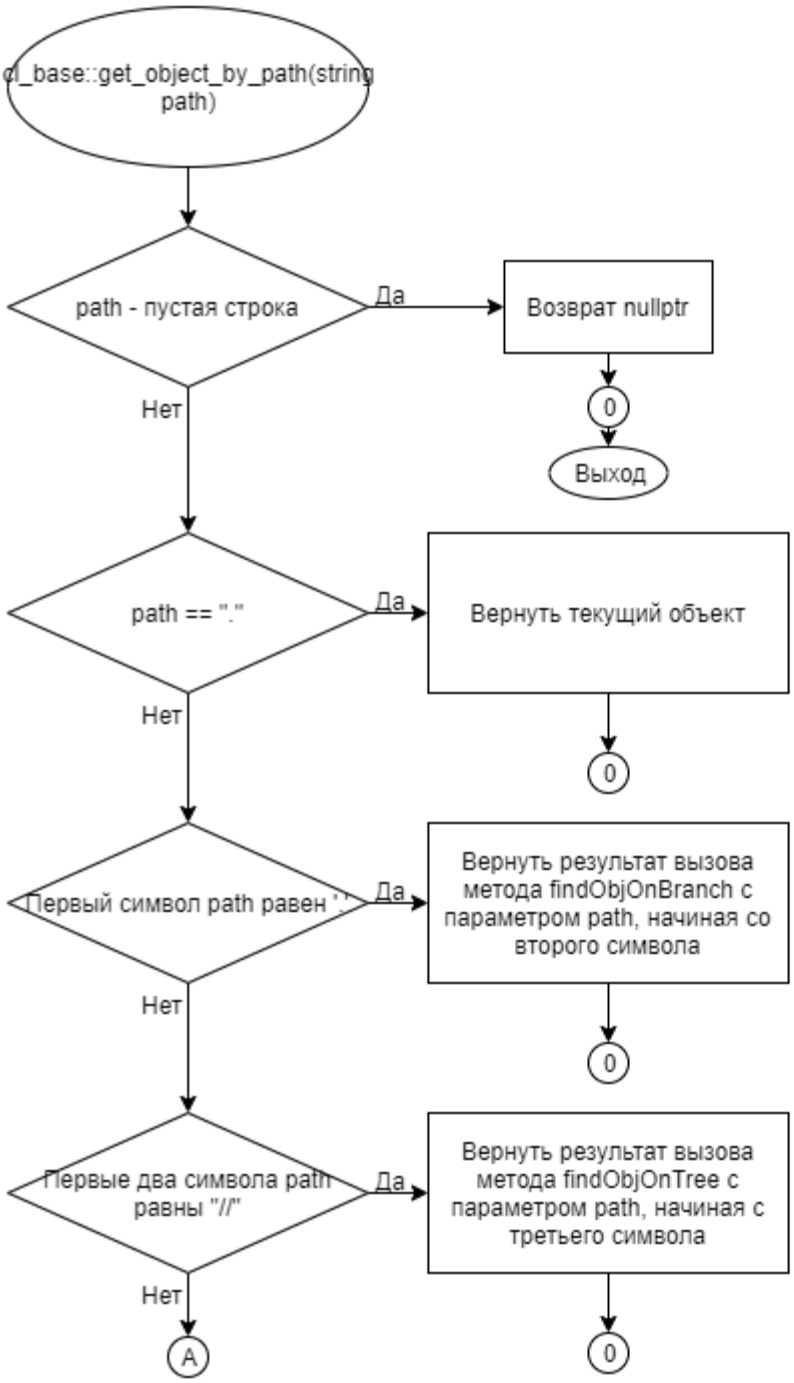

Рисунок 4 – Блок-схема алгоритма

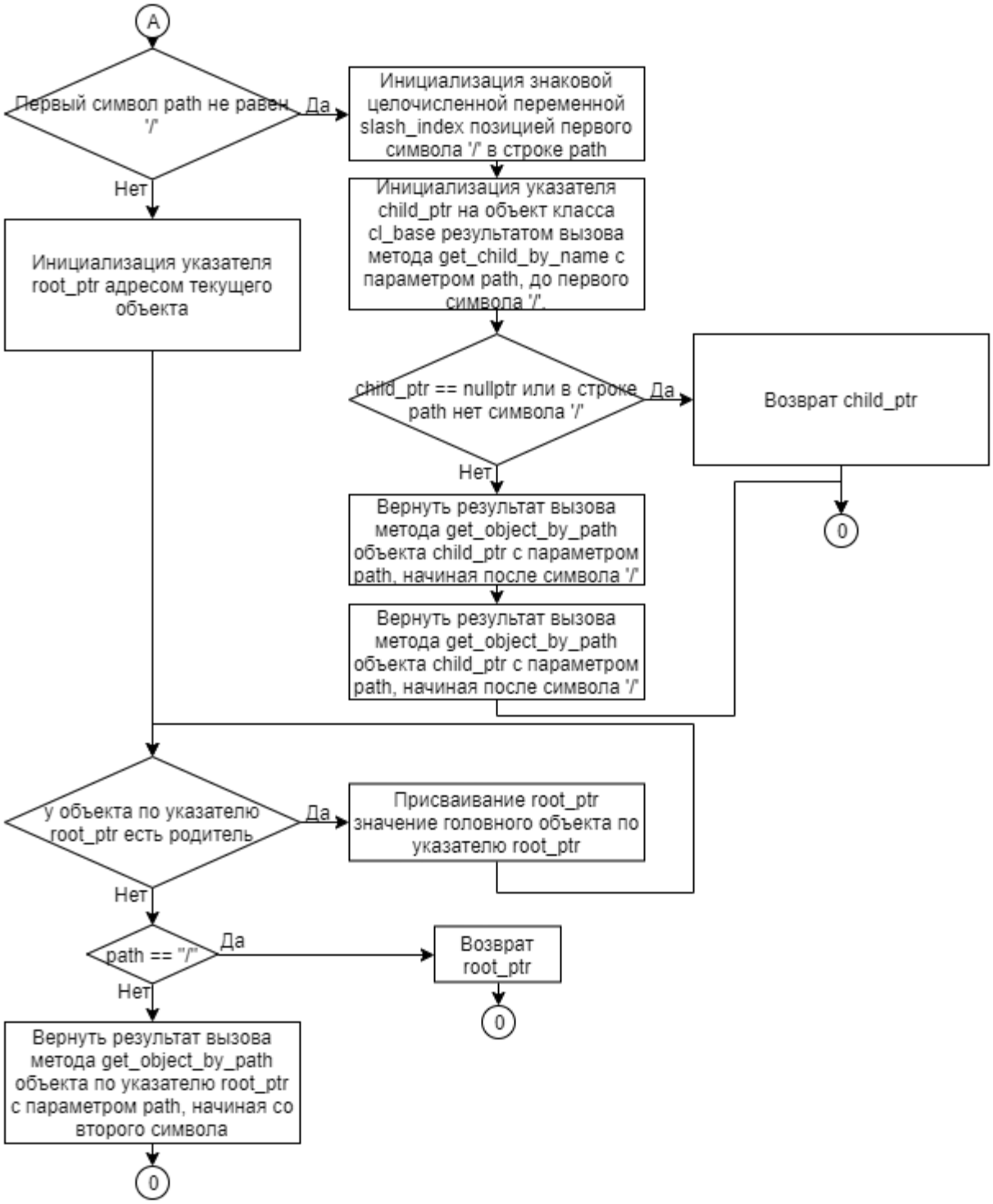

Рисунок 5 – Блок-схема алгоритма

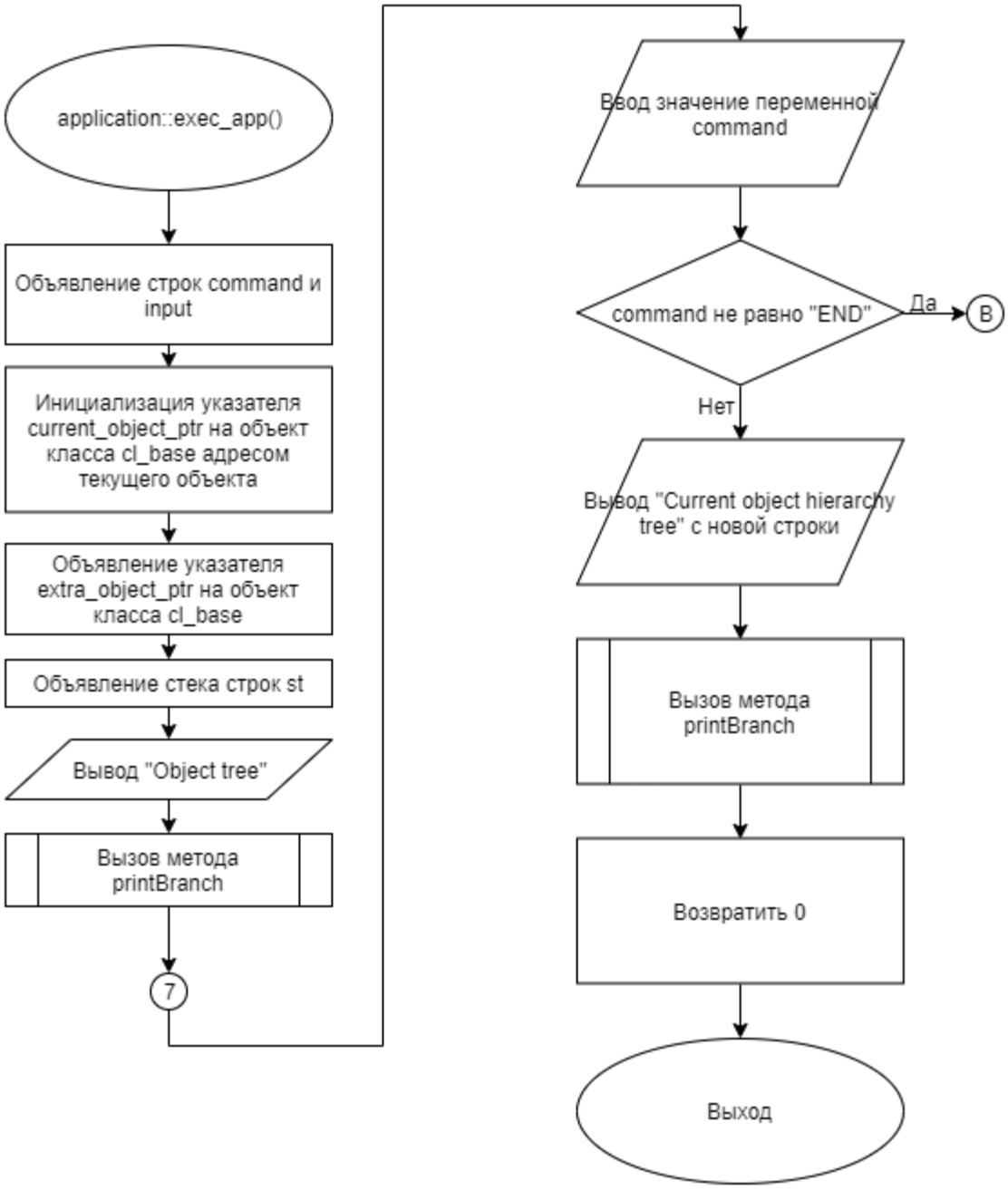

Рисунок 6 – Блок-схема алгоритма

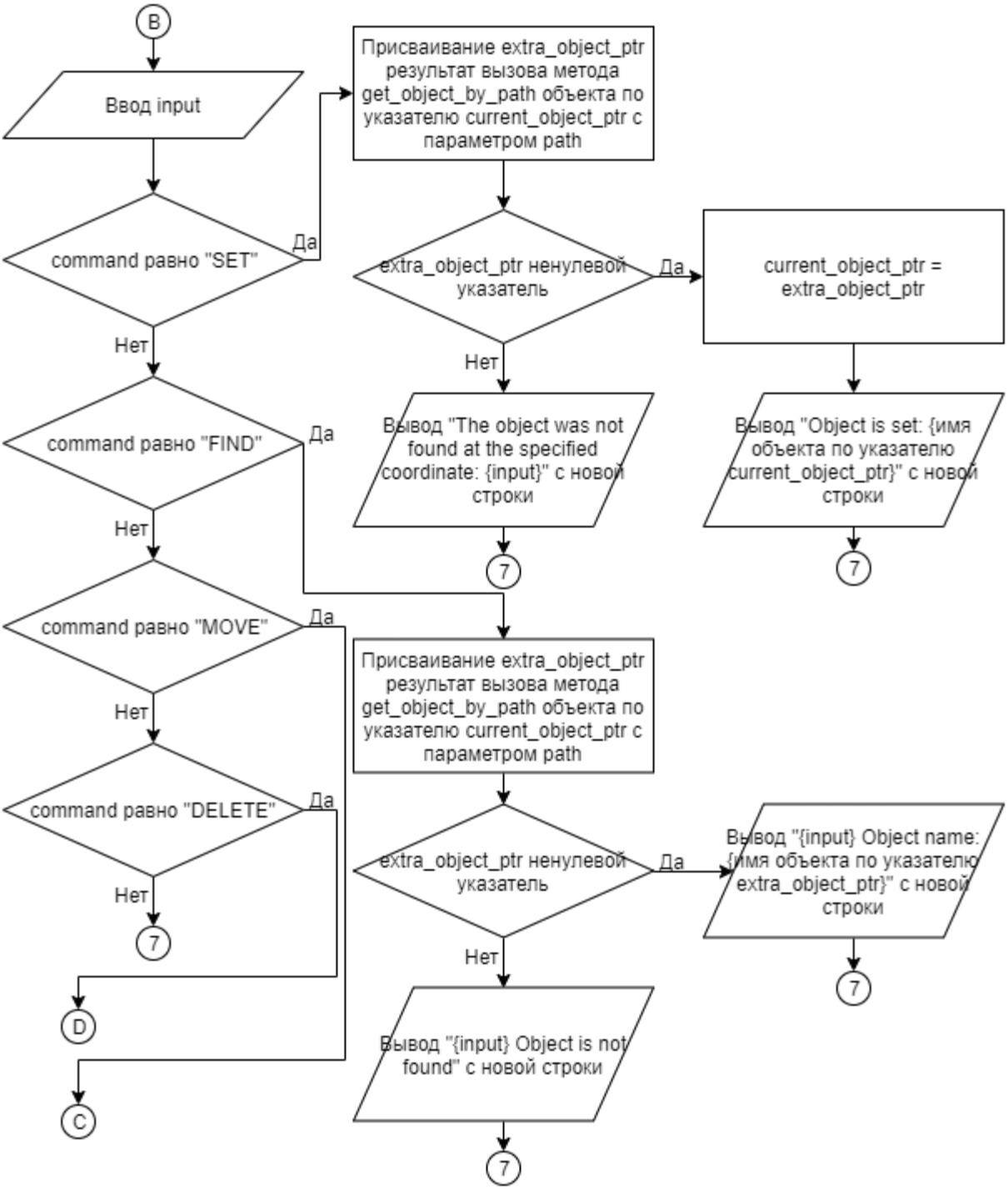

Рисунок 7 – Блок-схема алгоритма

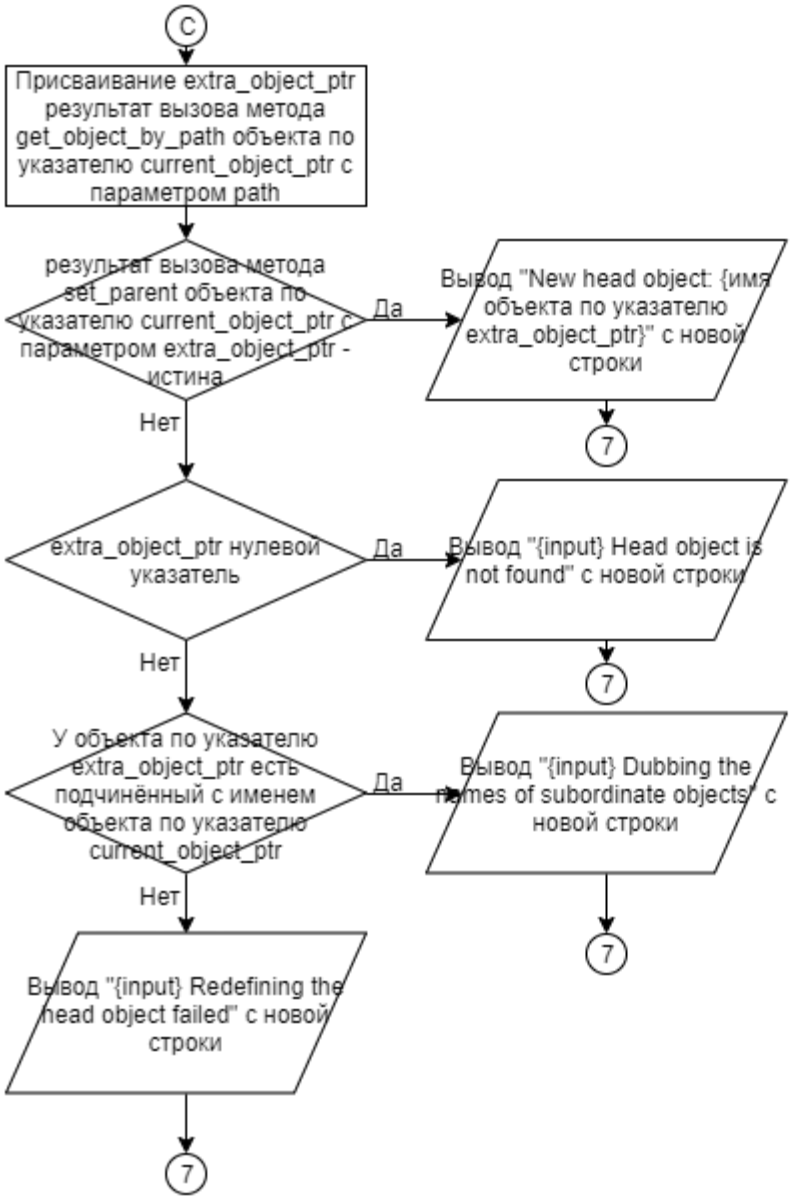

Рисунок 8 – Блок-схема алгоритма

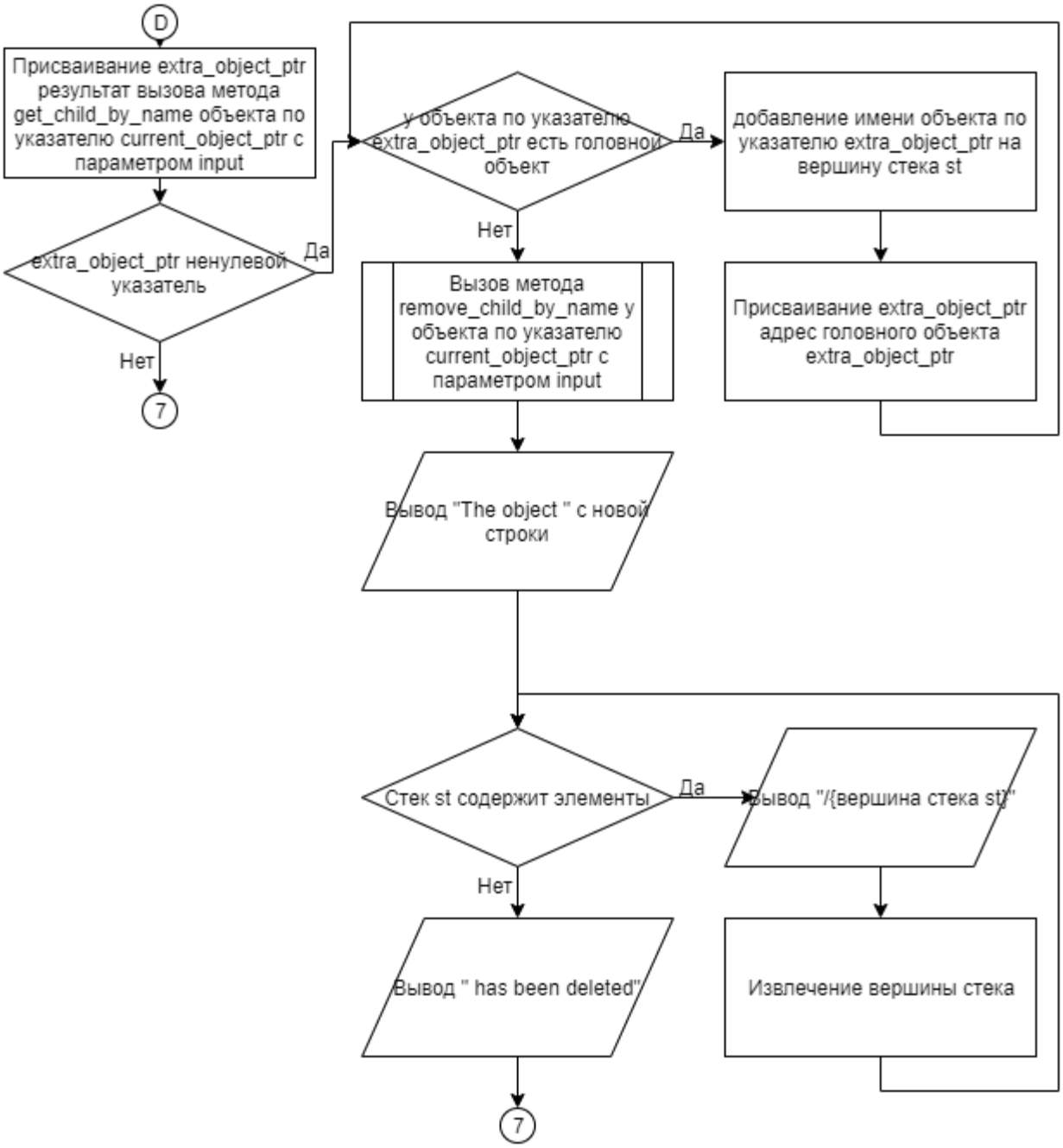

Рисунок 9 – Блок-схема алгоритма

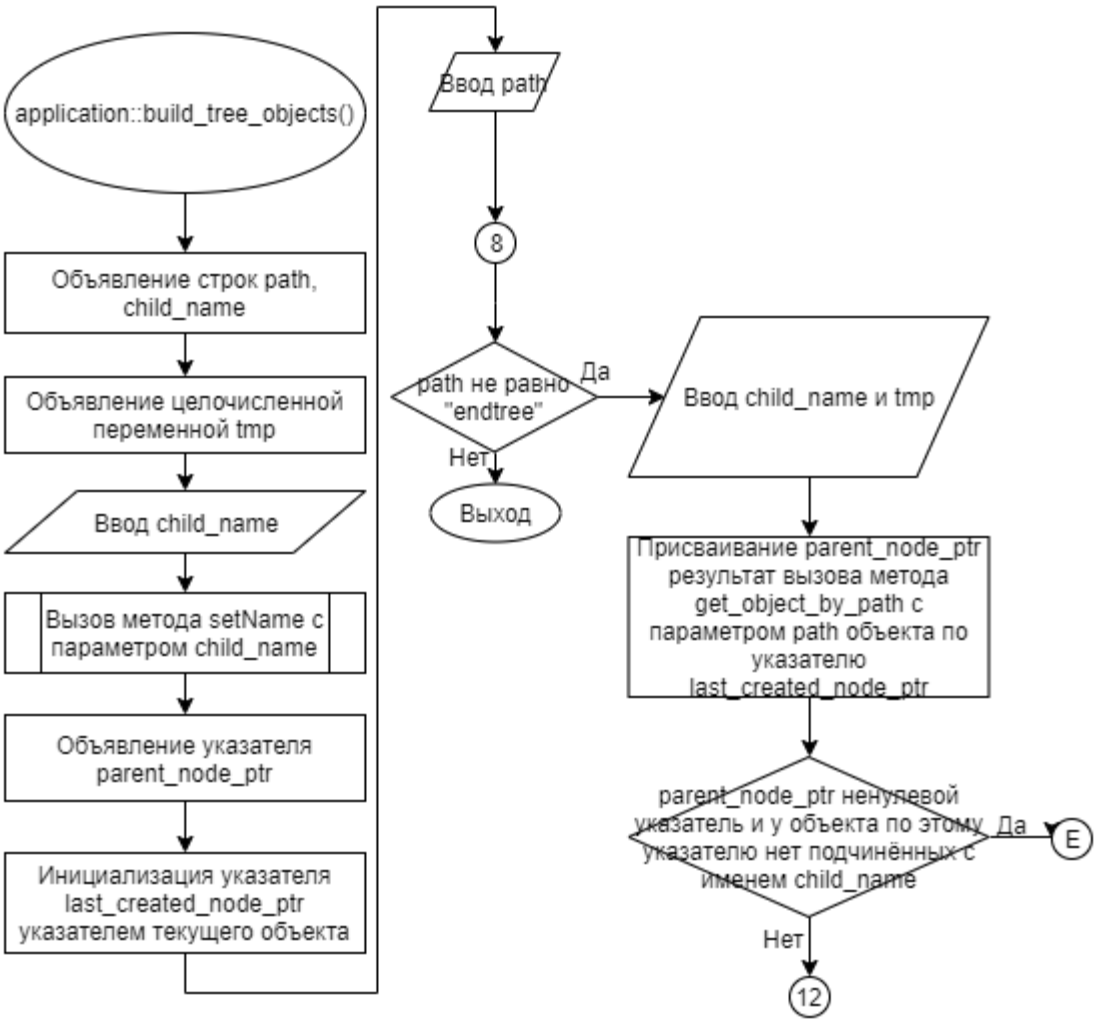

Рисунок 10 – Блок-схема алгоритма

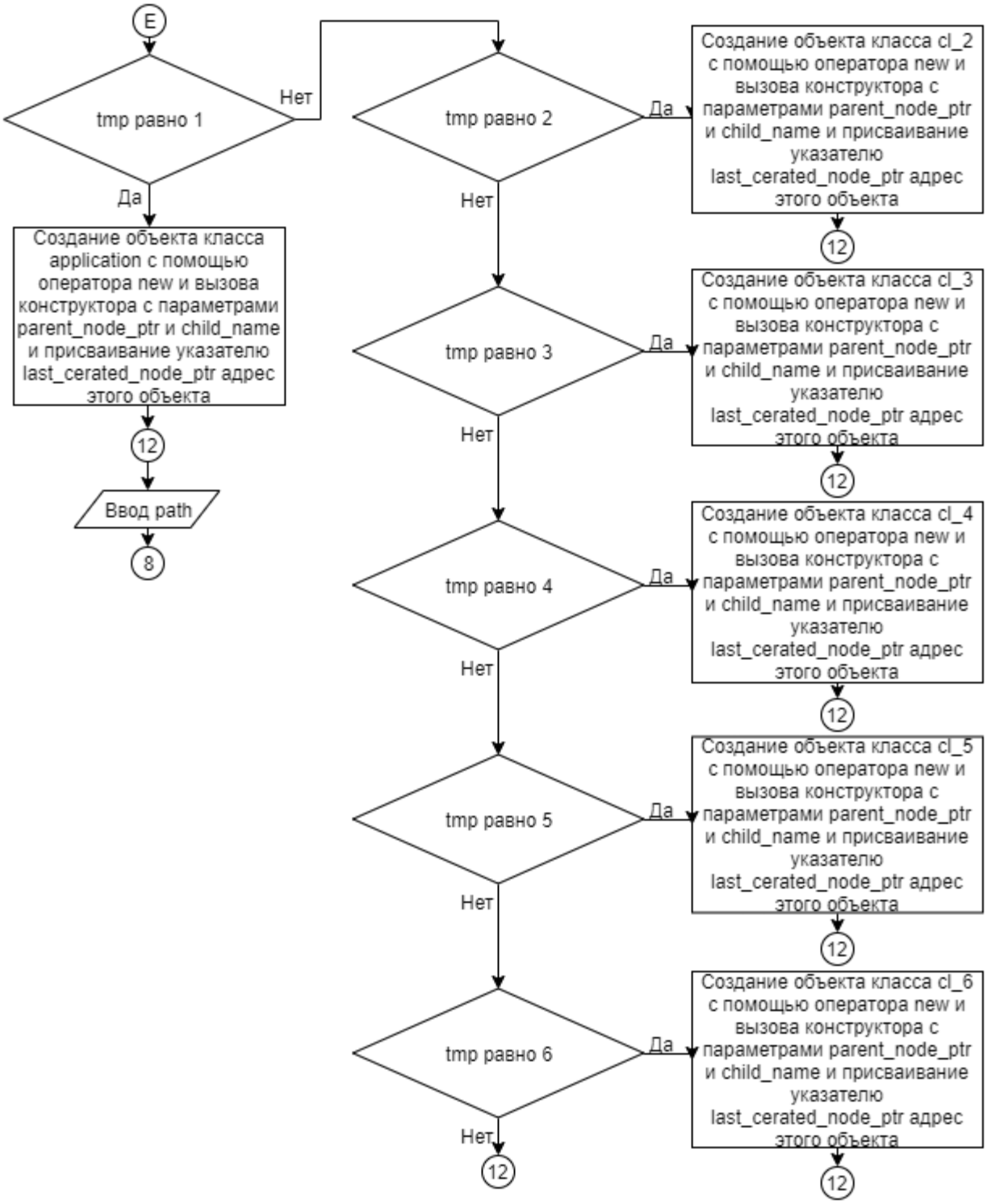

Рисунок 11 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл application.cpp

Листинг 1 – application.cpp

```
#include "application.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
#include <stack>

application::application(cl_base* parent): cl_base(parent) {}

int application::exec_app() {
      string command, input;
      cl_base* current_object_ptr = this;
      cl_base* extra_object_ptr;
      stack<string> st;
      this->printBranch();
      cin >> command;
      while (command != "END") {
            cin >> input;
            if (command == "SET") {
                  extra_object_ptr              =              current_object_ptr-
>get_object_by_path(input);
                  if (extra_object_ptr != nullptr) {
                        current_object_ptr = extra_object_ptr;
                        cout << endl << "Object is set: " << current_object_ptr-
>get_name();
                  }
                  else {
                        cout  <<  endl  <<  "The  object  was  not  found  at  the
specifiedcoordinate: " << input;
                  }
            }
            else if (command == "FIND") {
                  extra_object_ptr              =              current_object_ptr-
>get_object_by_path(input);
                  if (extra_object_ptr != nullptr) {
                        cout  <<  endl  <<  input  <<  "        Object  name:  "  <<
extra_object_ptr->get_name();
                  }
                  else {
                        cout << endl << input << "     Object is not found";
```

```
                  }
            }
            else if (command == "MOVE") {
                  extra_object_ptr              =              current_object_ptr-
>get_object_by_path(input);
                  if (current_object_ptr->set_parent(extra_object_ptr)) {
                        cout << endl << "New head object: " << extra_object_ptr-
>get_name();
                  }
                  else if (extra_object_ptr == nullptr) {
                        cout << endl << input << "     Head object is not found";
                  }
                  else if (extra_object_ptr->get_child_by_name(current_object_ptr-
>get_name()) != nullptr) {
                        cout << endl << input << "       Dubbing the names of
subordinate objects";
                  }
                  else {
                        cout << endl << input << "     Redefining the head object
failed";
                  }
            }
            else if (command == "DELETE") {
                  extra_object_ptr = current_object_ptr->get_child_by_name(input);
                  if (extra_object_ptr != nullptr) {
                        while (extra_object_ptr->get_parent() != nullptr) {
                              st.push(extra_object_ptr->get_name());
                              extra_object_ptr = extra_object_ptr->get_parent();
                        }
                        current_object_ptr->remove_child_by_name(input);
                        cout << endl << "The object ";
                        while (!st.empty()) {
                              cout << '/' << st.top();
                              st.pop();
                        }
                        cout << " has been deleted";
                  }
            }
            cin >> command;
      }
      cout << endl << "Current object hierarchy tree";
      this->printBranch();
      return 0;
}


void application::build_tree_objects() {
      cout << "Object tree";
      string path, child_name;
      int tmp;
      cin >> child_name;
      this->setName(child_name);
      cl_base* parent_node_ptr;
      cl_base* last_created_node_ptr = this;
      cin >> path;
      while (path != "endtree") {
            cin >> child_name >> tmp;
```

```
            parent_node_ptr = last_created_node_ptr->get_object_by_path(path);
            if (parent_node_ptr == nullptr) {
                  this->printBranch();
                  cout << endl << "The head object " << path << " is not found";
                  exit(1);
            }
            if (parent_node_ptr->get_child_by_name(child_name) != nullptr)
                  cout << endl << path << " Dubbing the names of subordinate
objects";
            else {
                  switch (tmp) {
                        case 1:
                              last_created_node_ptr              =              new
application(parent_node_ptr);
                              break;
                        case 2:
                              last_created_node_ptr  =  new  cl_2(parent_node_ptr,
child_name);
                              break;
                        case 3:
                              last_created_node_ptr  =  new  cl_3(parent_node_ptr,
child_name);
                              break;
                        case 4:
                              last_created_node_ptr  =  new  cl_4(parent_node_ptr,
child_name);
                              break;
                        case 5:
                              last_created_node_ptr  =  new  cl_5(parent_node_ptr,
child_name);
                              break;
                        case 6:
                              last_created_node_ptr  =  new  cl_6(parent_node_ptr,
child_name);
                              break;
                  }
            }
            cin >> path;
      }
}
```

5.2 Файл application.h

Листинг 2 – application.h

```
#ifndef __APPLICATION__H
#define __APPLICATION__H

#include "cl_base.h"
class application: public cl_base {
public:
      application(cl_base* parent);
      void build_tree_objects();
```

```
	int exec_app();
};


#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
{
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 {
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 {
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 {
public:
      cl_5(cl_base* p_head_object, string s_object_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_object_name):cl_base(p_head_object,
s_object_name){

}
```

5.12 Файл cl_6.h

Листинг 12 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
```

```
#include "cl_base.h"
class cl_6: public cl_base {
public:
	cl_6(cl_base* p_head_object, string s_object_name);
};

#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
#include <stack>

cl_base::cl_base(cl_base* parent, string name): parent(parent), name(name) {
	if (parent != nullptr) {
		parent->children.push_back(this);
	}
}
cl_base::~cl_base() {
	for (int i = 0; i < children.size(); i++) delete children[i];
}

string cl_base::get_name() const {return name;}
cl_base* cl_base::get_parent() const {return parent;}

bool cl_base::setName(string name1) {
	if (get_parent() != nullptr && get_parent() -> get_child_by_name(name1) !=
nullptr) {
		return false;
	}
	name = name1;
	return true;
}

cl_base* cl_base::get_child_by_name(string name) {
	for (auto child: children) {
		if (child->name == name) return child;
	}
	return nullptr;
}

//2 часть

cl_base* cl_base::findObjOnBranch(string s_object_name) {
	cl_base* found = nullptr;
	queue <cl_base*> elementsQueue;
	elementsQueue.push(this);
	while(!elementsQueue.empty()) {
		cl_base* elem = elementsQueue.front();
		elementsQueue.pop();
```

```
            if (elem->name == s_object_name) {
                  if (found != nullptr) {
                        return nullptr;
                  }
                  else {
                        found = elem;
                  }
            }
            for (int i = 0; i < elem->children.size();i++) {
                  elementsQueue.push(elem->children[i]);
            }
      }
      return found;
}
cl_base* cl_base::findObjOnTree(string s_object_name) {
      if (parent != nullptr) {
            return parent->findObjOnTree(s_object_name);
      }
      else {
            return findObjOnBranch(s_object_name);
      }
}
void cl_base::printBranch(int level) {
      cout << endl;
      for (int i = 0; i < level; ++i) {
            cout << "    ";
      }
      cout << this->get_name();
      for (int i = 0; i < children.size(); ++i) {
            children[i]->printBranch(level + 1);
      }
}
void cl_base::printBranchWithState(int level) {
      cout << endl;
      for (int i = 0; i < level; ++i) {
            cout << "    ";
      }
      if (this->state != 0) {
            cout << this->get_name() << " is ready";
      }
      else {
            cout << this->get_name() << " is not ready";
      }
      for (int i= 0; i < children.size(); ++i) {
            children[i]->printBranchWithState(level + 1);
      }
}
void cl_base::setState(int state) {
      if (parent == nullptr || parent->state != 0) {
            this->state = state;
      }
      if (state == 0) {
            this->state = state;
            for (int i = 0; i < children.size(); i++) {
                  children[i]->setState(state);
            }
      }
```

```
}

//3 часть

bool cl_base::set_parent(cl_base* new_parent) {
	if (this->get_parent() == new_parent) {
		return true;
	}
	if (this->get_parent() == nullptr || new_parent == nullptr) {
		return false;
	}
	if (new_parent->get_child_by_name(this->get_name()) != nullptr) {
		return false;
	}
	stack<cl_base*> st;
	st.push(this);
	while (!st.empty()) {
		cl_base* current_node_ptr = st.top();
		st.pop();
		if (current_node_ptr == new_parent) {
			return false;
		}
		for (int i = 0; i < current_node_ptr->children.size(); ++i) {
			st.push(current_node_ptr->children[i]);
		}
	}
	vector<cl_base*> & v = this->get_parent()->children;
	for (int i = 0; i < v.size(); ++i) {
		if (v[i]->get_name() == this->get_name()) {
			v.erase(v.begin() + i);
			new_parent->children.push_back(this);
			return true;
		}
	}
	return false;
}
void cl_base::remove_child_by_name(string child_name) {
	vector<cl_base*> & v = this->children;
	for (int i = 0; i < v.size(); ++i) {
		if (v[i]->get_name() == child_name) {
			delete v[i];
			v.erase(v.begin() + i);
			return;
		}
	}
}
cl_base* cl_base::get_object_by_path(string path) {
	if (path.empty()) {
		return nullptr;
	}
	if (path == ".") {
		return this;
	}
	if (path[0] == '.') {
		return findObjOnBranch(path.substr(1));
	}
	if (path.substr(0, 2) == "//") {
```

```
            return this->findObjOnTree(path.substr(2));
      }
      if (path[0] != '/') {
            size_t slash_index = path.find('/');
            cl_base*     child_ptr     =     this->get_child_by_name(path.substr(0,
slash_index));
            if (child_ptr == nullptr || slash_index == string::npos) {
                  return child_ptr;
            }
            return child_ptr->get_object_by_path(path.substr(slash_index + 1));
      }
      cl_base* root_ptr = this;
      while (root_ptr->get_parent() != nullptr) {
            root_ptr = root_ptr->get_parent();
      }
      if (path == "/") {
            return root_ptr;
      }
      return root_ptr->get_object_by_path(path.substr(1));
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H

#include <iostream>
#include <vector>
#include <string>
#include <queue>
using namespace std;
class cl_base {
private:
      int state = 0;
      cl_base* parent;
      vector <cl_base*> children;
      string name;
public:
      cl_base(cl_base* parent, string name = "Object_root");
      ~cl_base();
      bool setName(string name);
      string get_name() const;
      cl_base* get_parent() const;
      cl_base* get_child_by_name(string name);
      //2 часть
      cl_base* findObjOnBranch(string name);
      cl_base* findObjOnTree(string name);
      void printBranch(int level = 0);
      void printBranchWithState(int level = 0);
      void setState(int state);
      //3 часть
```

```
      bool set_parent(cl_base* new_parent);
      void remove_child_by_name(string child_name);
      cl_base* get_object_by_path(string path);
};


#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

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

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 7.

Таблица 7 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

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
	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
		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

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).