Здесь будет титульник, листай ниже

**СОДЕРЖАНИЕ**

[1 ПОСТАНОВКА ЗАДАЧИ 6](#__RefHeading___Toc9984_1997490348)

[1.1 Описание входных данных 8](#__RefHeading___Toc9986_1997490348)

[1.2 Описание выходных данных 9](#__RefHeading___Toc9988_1997490348)

[2 МЕТОД РЕШЕНИЯ 12](#__RefHeading___Toc9990_1997490348)

[3 ОПИСАНИЕ АЛГОРИТМОВ 13](#__RefHeading___Toc9992_1997490348)

[3.1 Алгоритм функции main 13](#__RefHeading___Toc9994_1997490348)

[3.2 Алгоритм метода build\_tree\_objects класса cl\_application 13](#__RefHeading___Toc9996_1997490348)

[3.3 Алгоритм метода exec\_app класса cl\_application 17](#__RefHeading___Toc9998_1997490348)

[3.4 Алгоритм метода CheckingObjUniq класса cl\_base 17](#__RefHeading___Toc10000_1997490348)

[3.5 Алгоритм метода SearchObjOnTree класса cl\_base 18](#__RefHeading___Toc10002_1997490348)

[3.6 Алгоритм метода GetRoot класса cl\_base 18](#__RefHeading___Toc10004_1997490348)

[3.7 Алгоритм метода GetObjectByPath класса cl\_base 19](#__RefHeading___Toc10006_1997490348)

[3.8 Алгоритм метода ChangeHeadObj класса cl\_base 20](#__RefHeading___Toc10008_1997490348)

[3.9 Алгоритм метода DeleteSubObj класса cl\_base 21](#__RefHeading___Toc10010_1997490348)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 23](#__RefHeading___Toc10012_1997490348)

[5 КОД ПРОГРАММЫ 33](#__RefHeading___Toc10014_1997490348)

[5.1 Файл cl\_1.cpp 33](#__RefHeading___Toc10016_1997490348)

[5.2 Файл cl\_1.h 33](#__RefHeading___Toc10018_1997490348)

[5.3 Файл cl\_2.cpp 33](#__RefHeading___Toc10020_1997490348)

[5.4 Файл cl\_2.h 34](#__RefHeading___Toc10022_1997490348)

[5.5 Файл cl\_3.cpp 34](#__RefHeading___Toc10024_1997490348)

[5.6 Файл cl\_3.h 34](#__RefHeading___Toc10026_1997490348)

[5.7 Файл cl\_4.cpp 34](#__RefHeading___Toc10028_1997490348)

[5.8 Файл cl\_4.h 35](#__RefHeading___Toc10030_1997490348)

[5.9 Файл cl\_5.cpp 35](#__RefHeading___Toc10032_1997490348)

[5.10 Файл cl\_5.h 35](#__RefHeading___Toc10034_1997490348)

[5.11 Файл cl\_6.cpp 35](#__RefHeading___Toc10036_1997490348)

[5.12 Файл cl\_6.h 36](#__RefHeading___Toc10038_1997490348)

[5.13 Файл cl\_application.cpp 36](#__RefHeading___Toc10040_1997490348)

[5.14 Файл cl\_application.h 39](#__RefHeading___Toc10042_1997490348)

[5.15 Файл cl\_base.cpp 40](#__RefHeading___Toc10044_1997490348)

[5.16 Файл cl\_base.h 44](#__RefHeading___Toc10046_1997490348)

[5.17 Файл main.cpp 45](#__RefHeading___Toc10048_1997490348)

[6 ТЕСТИРОВАНИЕ 46](#__RefHeading___Toc10050_1997490348)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#__RefHeading___Toc10052_1997490348)

# 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

Для решения задачи используется то же, что и в работе КВ2, однако был использован поток чтения/записи данных - stringstream, и изменены/добавлены несколько методов

Класс cl\_base:

* Свойства/поля:
  + Те же поля, что и в КВ2 у класса cl\_base;
* Методы
  + Те же методы, что и в КВ2 у класса cl\_base;
  + Метод CheckingObjUniq
    - Функционал - используется для определения уникальности наименования объекта на ветке дерева иерархии;
  + Метод GetRoot
    - Функционал - используется для получения корневого объекта дерева иерархии
  + Метод GetObjectByPath;
    - Функционал - используется для получения указателя на любой объект в дереве иерархии согласно пути;
  + Метод ChangeHeadObj
    - Функционал - используется для переопределения головного объекта для текущего в дереве иерархии;
  + Метод DeleteSubObj
    - Функционал - используется для удаления подчиненного объекта по наименованию;

# 3 ОПИСАНИЕ АЛГОРИТМОВ

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

## Алгоритм функции main

Функционал: Основной алгоритм работы программы.

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения программы.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Выполнение алгоритма данной функции из предыдущей работы - КВ2 | ∅ |

## Алгоритм метода build\_tree\_objects класса cl\_application

Функционал: Построение исходного дерева иерархии объектов.

Параметры: Отсутствуют.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода build\_tree\_objects класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление строковых переменных objPath, childName, command и целочисленной переменной classNum | 2 |
| 2 |  | Инициализация указателя currentObj на объект класса cl\_base значением this | 3 |
| 3 |  | Ввод значения objPath | 4 |
| 4 |  | Вызов метода SetName с аргументом objPath | 5 |
| 5 |  | Очистка потока ввода cin | 6 |
| 6 |  | Ввод значения objPath | 7 |
| 7 | objPath равен "endtree" | Выход из цикла | 14 |
|  | Инициализация указателя parentObj на объект класса cl\_base результатом выполнения метода GetObjectByPath с аргументом objPath | 8 |
| 8 | parentObj равен nullptr | Вывод "Object tree" на экран | 9 |
|  | Ввод значений childName и classNum | 12 |
| 9 |  | Вызов метода PrintObjects текущего объекта | 10 |
| 10 |  | Вывод "\nThe head object ", objPath, " is not found" на экран | 11 |
| 11 |  | Прекращение работы программы с кодом возврата 1 | ∅ |
| 12 | parentObj не имеет подчиненного объекта с наименованием childName |  | 13 |
|  | Вывод objPath и " Dubbing the names of subordinate objects" на экран | 5 |
| 13 | Значение classNum равно i-тому значению | Создание объекта cl\_i-того класса с параметрами parentObj и childName | 5 |
|  |  | 5 |
| 14 |  | Вывод "Object tree" на экран | 15 |
| 15 |  | Вызов метода PrintObjects текущего объекта | 16 |
| 16 |  | Переход на новую строку | 17 |
| 17 |  | Ввод значения command | 18 |
| 18 | command равен "END" | Выход из цикла | ∅ |
|  | Ввод значения objPath | 19 |
| 19 |  | Инициализация указателя obj на объект класса cl\_base результатом выполнения метода GetObjectByPath объекта currentObj с аргументом objPath | 20 |
| 20 | command равен "SET" |  | 21 |
| command равен "FIND" |  | 23 |
| command равен "MOVE" |  | 24 |
| command равен "DELETE" |  | 25 |
|  |  | 17 |
| 21 | obj не равен nullptr | Присваивание currentObj значение obj | 22 |
|  | Вывод "The object was not found at the specified coordinate: " и objPath на экран | 17 |
| 22 |  | Вывод "Object is set: " и наименование obj на экран | 17 |
| 23 | obj не равен nullptr | Вывод objPath, " Object name: " и наименование obj на экран | 17 |
|  | Вывод objPath и " Object is not found" на экран | 17 |
| 24 | obj равен nullptr | Вывод objPath и " Head object is not found" на экран | 17 |
| Родители obj и currentObj не равны и currentObj имееет на ветке объект с именем объекта obj | Вывод objPath и " Redefining the head object  failed" на экран | 17 |
| obj имеет подчиненный объект с именем объекта currentObj | Вывод objPath и " Dubbing the names of subordinate objects" на экран | 17 |
| Смена головного объекта currentObj не получилась | Вывод objPath и " Redefining the head object failed" на экран | 17 |
|  | Вывод "New head object: " и наименование obj на экран | 17 |
| 25 | obj не равен nullptr | Инициализация указателя temp на объект класса cl\_base значением указателя на головной объект объекта obj | 26 |
|  |  | 17 |
| 26 |  | Объявление строковой переменной ans | 27 |
| 27 | Объект с именем объекта obj удален из currentObj |  | 28 |
|  |  | 17 |
| 28 | temp не равен nullptr |  | 29 |
|  | Вывод "The object ", ans + "/" + objPath и " has been deleted" на экран | 17 |
| 29 | Указатель на головной объект объекта temp не равен nullptr | Присваивание к значению ans - "/" и наименование temp | 30 |
|  |  | 30 |
| 30 |  | Присваивание temp значение головного объекта temp | 28 |

## Алгоритм метода exec\_app класса cl\_application

Функционал: Запуск приложения (начало функционирования системы).

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения метода.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода exec\_app класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод "Current object hierarchy tree" на экран | 2 |
| 2 |  | Вызов метода PrintObjects() текущего объекта | 3 |
| 3 |  | Возврат значения 0 | ∅ |

## Алгоритм метода CheckingObjUniq класса cl\_base

Функционал: Поиск объекта на ветке, проверяя при этом его уникальность.

Параметры: string objName - наименование искомого объекта.

Возвращаемое значение: cl\_base\* - указатель на искомый объект.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода CheckingObjUniq класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Результат метода ObjNameCount с аргументом objName не равен 1 | Возврат nullptr | ∅ |
|  | Возврат результата метода SearchObjOnBranch с аргументом objName | ∅ |

## Алгоритм метода SearchObjOnTree класса cl\_base

Функционал: Поиск объекта на всем дереве иерархии.

Параметры: string objName - наименование искомого объекта.

Возвращаемое значение: cl\_base\* - указатель на искомый объект.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода SearchObjOnTree класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Возврат результата метода CheckingObjUniq с аргументом objName, вызванного для результата метода GetRoot текущего объекта | ∅ |

## Алгоритм метода GetRoot класса cl\_base

Функционал: Получение корневого объекта дерева иерархии.

Параметры: Отсутствуют.

Возвращаемое значение: cl\_base\* - указатель на корневой объект дерева иерархии.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода GetRoot класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация указателя obj на объект класса cl\_base значением this | 2 |
| 2 | Указатель на головной объект объекта temp не равен nullptr | Присваивание obj значение головного объекта obj | 2 |
|  | Возврат obj | ∅ |

## Алгоритм метода GetObjectByPath класса cl\_base

Функционал: Получение указателя на любой объект в дереве иерархии согласно пути path.

Параметры: string path - путь поиска указателя на объект.

Возвращаемое значение: cl\_base\* - указатель на искомый объект.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода GetObjectByPath класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация указателя currentObj на объекткласса cl\_base значением this | 2 |
| 2 |  | Объявление строковой переменной nextObjName | 3 |
| 3 | Первый символ path равен "/" |  | 4 |
| Первый символ path равен "." |  | 6 |
|  |  | 7 |
| 4 | Второй символ path равен "/" | Возврат результата метода SearchObjOnTree с аргументом path без 2-х первых символов | ∅ |
|  | Присваивание currentObj результат метода GetRoot | 5 |
| 5 |  | Присваивание path значение path без первого элемента | 7 |
| 6 | path равен "." | Возврат currentObj | ∅ |
|  | Возврат результата метода CheckingObjUniq с аргументом path без первого символа | ∅ |
| 7 |  | Создание объекта streamPath типа stringstream, с передачей в него параметра path | 8 |
| 8 | Можно разбивать streamPath на подстроки, используя '/' как разделитель | Присваивание nextObjName значение найденной подстроки | 9 |
|  | Возврат значения currentObj | ∅ |
| 9 |  | Присваивание currentObj результат метода GetChild c аргументом nextObjName объекта currentObj | 10 |
| 10 | currentObj равен nullptr | Возврат nullptr | ∅ |
|  |  | 8 |

## Алгоритм метода ChangeHeadObj класса cl\_base

Функционал: Переопределение головного объекта для текущего в дереве иерархии.

Параметры: cl\_base\* newHead - указатель на новый головной объект текущего.

Возвращаемое значение: bool - индикатор корректности выполнения метода.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода ChangeHeadObj класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Указатель на головной объект текущего объекта не равен nullptr | Инициализация итератора i значением итератора первого элемента списка children головного объекта | 2 |
|  | Возврат false | ∅ |
| 2 | Значение i не равно значению итератора элемента, следующего за конечным |  | 3 |
|  |  | 4 |
| 3 | Объект, на который указывает i, равен this | Вызов метода erase для списка children головного объекта, с передачей в него итератора i, и выход из цикла | 4 |
|  | Инкремент значения итератора i | 2 |
| 4 |  | Присваивание полю parent текущего объекта значение параметра newHead | 5 |
| 5 |  | Добавление в конец списка children головного объекта указатель на текущий объект | 6 |
| 6 |  | Возврат true | ∅ |

## Алгоритм метода DeleteSubObj класса cl\_base

Функционал: Удаление подчиненного объекта по наименованию.

Параметры: string objName - наименование удаляемого объекта.

Возвращаемое значение: bool - индикатор корректности выполнения метода.

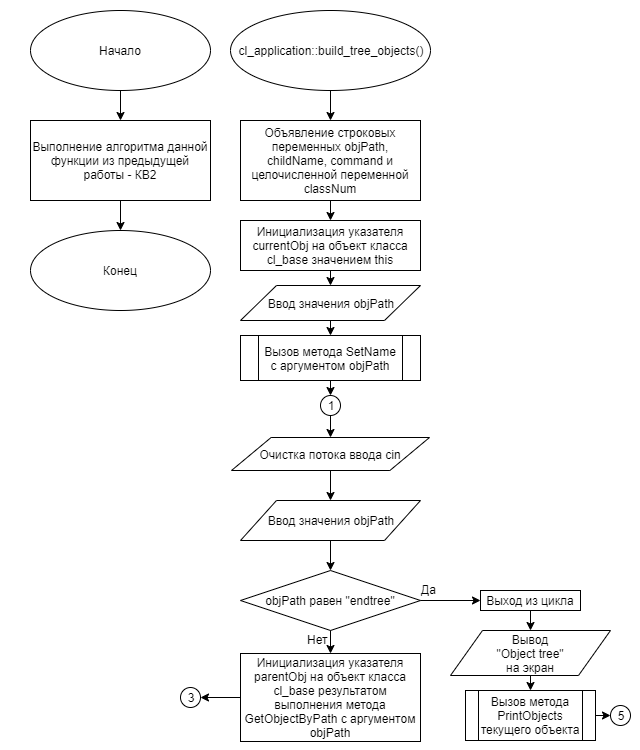
Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода DeleteSubObj класса cl\_base

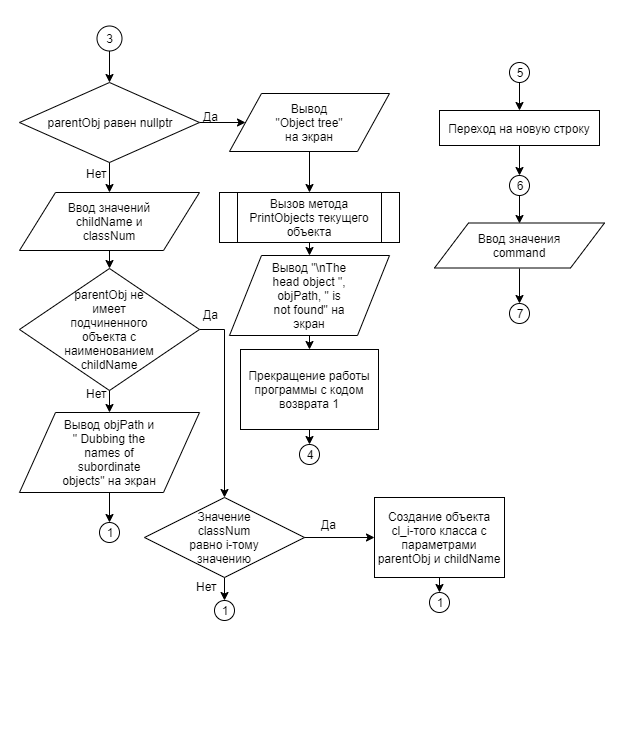
| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация указателя subObj на объект класса cl\_base значением метода GetChild с аргументом objName | 2 |
| 2 |  | Инициализация итератора i значением итератора первого элемента списка children | 3 |
| 3 | Значение i не равно значению итератора элемента, следующего за конечным |  | 4 |
|  | Возврат false | ∅ |
| 4 | Объект, на который указывает i, равен subObj | Вызов метода erase для списка children, с передачей в него итератора i | 5 |
|  | Инкремент значения итератора i | 3 |
| 5 |  | Удаление subObj из памяти через оператор delete | 6 |
| 6 |  | Возврат true | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

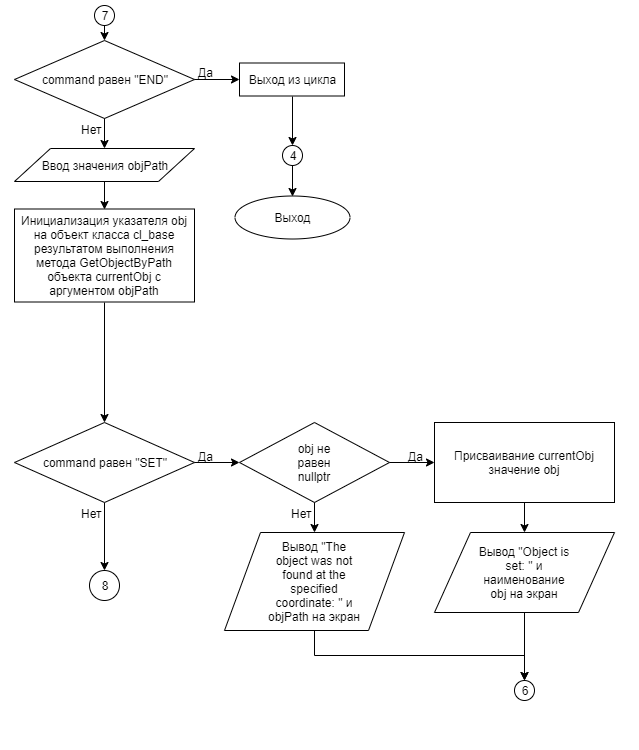
Представим описание алгоритмов в графическом виде на рисунках 1-10.



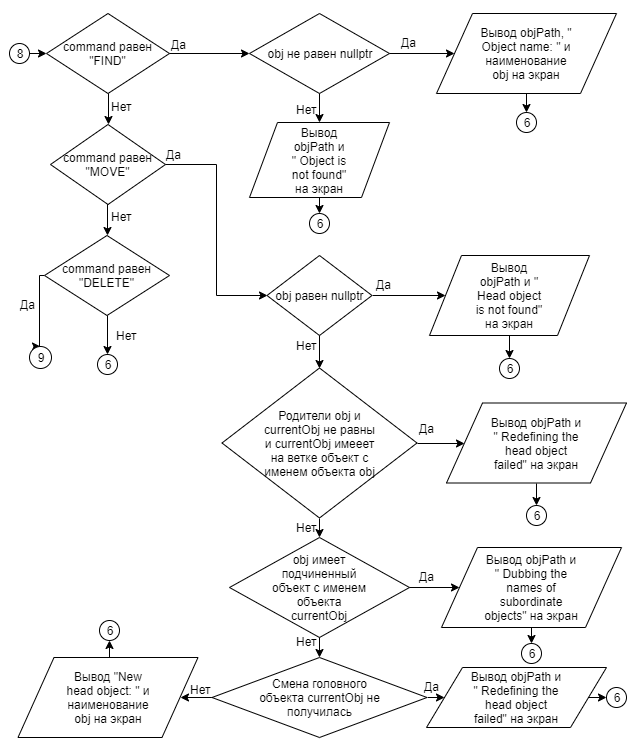
**Рисунок 1 – Блок-схема алгоритма**



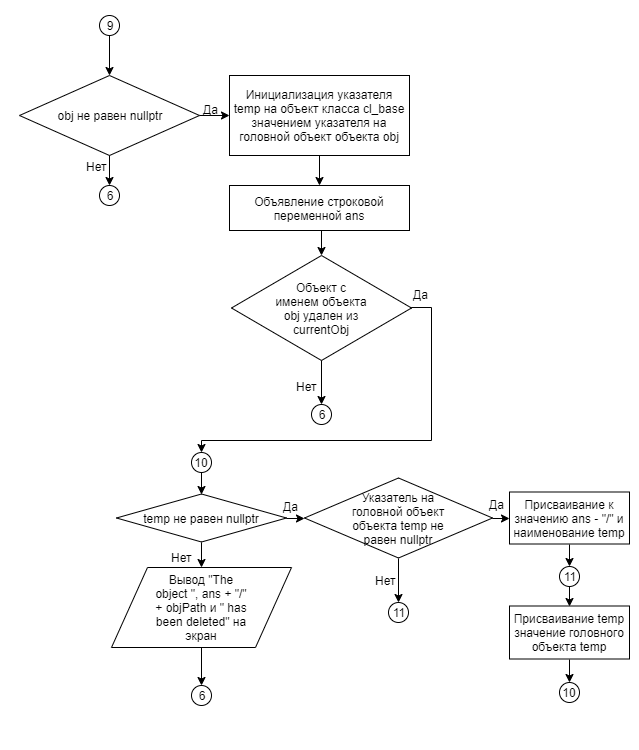
**Рисунок 2 – Блок-схема алгоритма**



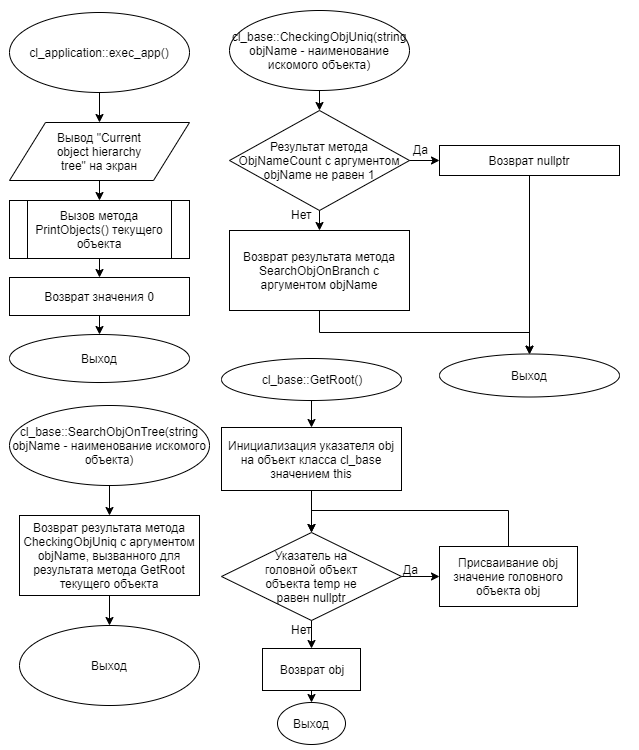
**Рисунок 3 – Блок-схема алгоритма**



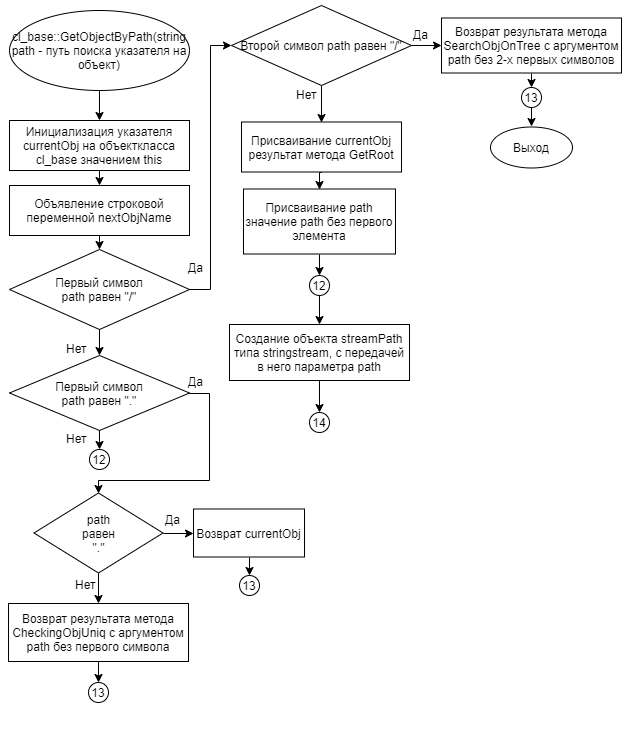
**Рисунок 4 – Блок-схема алгоритма**



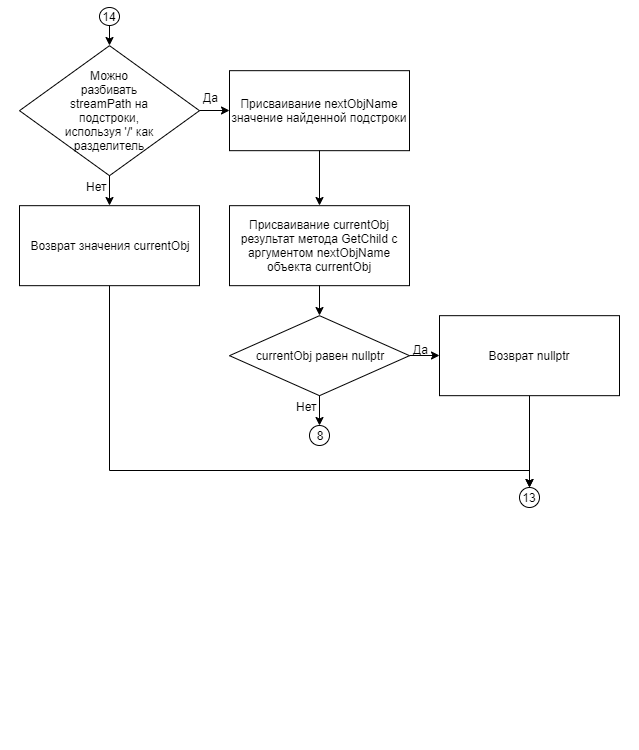
**Рисунок 5 – Блок-схема алгоритма**



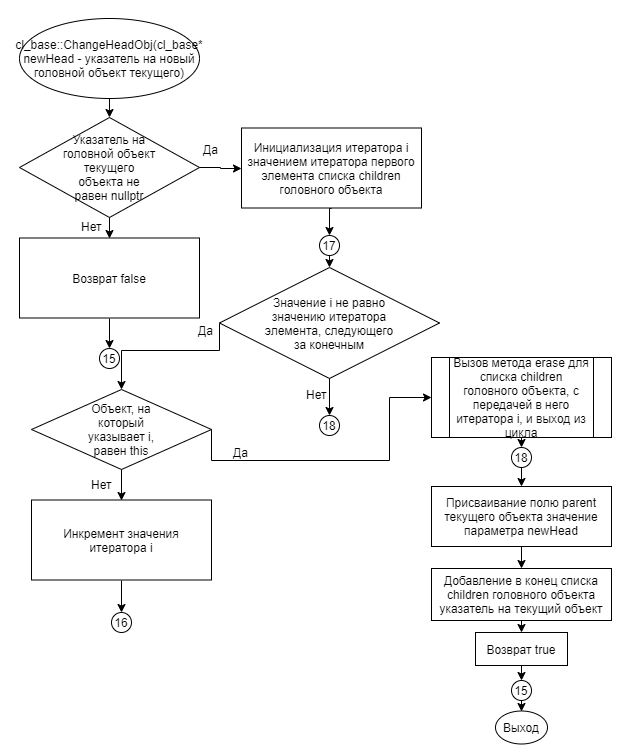
**Рисунок 6 – Блок-схема алгоритма**



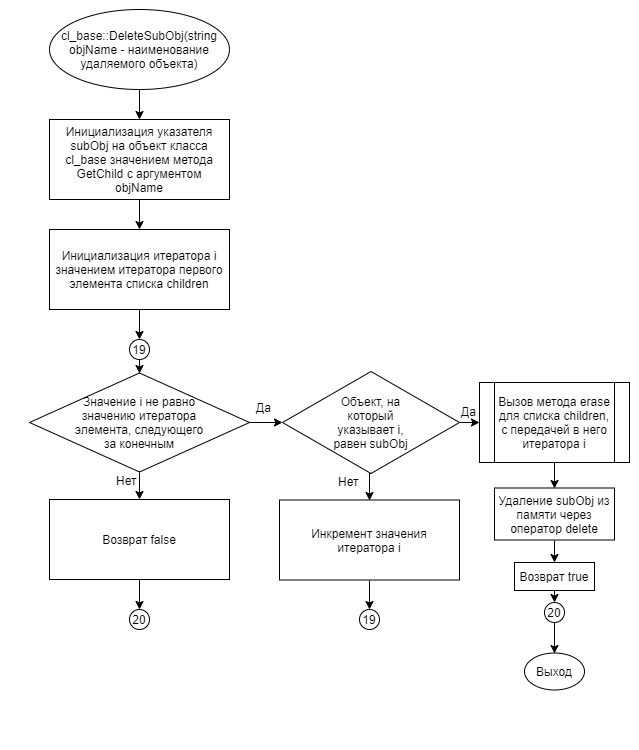
**Рисунок 7 – Блок-схема алгоритма**



**Рисунок 8 – Блок-схема алгоритма**



**Рисунок 9 – Блок-схема алгоритма**



**Рисунок 10 – Блок-схема алгоритма**

# 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

## **Файл** **cl\_1.cpp**

Листинг 1 – cl\_1.cpp

|  |
| --- |
| #include "cl\_1.h"  cl\_1::cl\_1(cl\_base\* parent, string name) : cl\_base(parent, name)  {} |

## **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\* parent, string name);  };  #endif |

## **Файл** **cl\_2.cpp**

Листинг 3 – cl\_2.cpp

|  |
| --- |
| #include "cl\_2.h"  cl\_2::cl\_2(cl\_base\* parent, string name) : cl\_base(parent, name)  {} |

## **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\* parent, string name);  };  #endif |

## **Файл** **cl\_3.cpp**

Листинг 5 – cl\_3.cpp

|  |
| --- |
| #include "cl\_3.h"  cl\_3::cl\_3(cl\_base\* parent, string name) : cl\_base(parent, name)  {} |

## **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\* parent, string name);  };  #endif |

## **Файл** **cl\_4.cpp**

Листинг 7 – cl\_4.cpp

|  |
| --- |
| #include "cl\_4.h"  cl\_4::cl\_4(cl\_base\* parent, string name) : cl\_base(parent, name)  {} |

## **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\* parent, string name);  };  #endif |

## **Файл** **cl\_5.cpp**

Листинг 9 – cl\_5.cpp

|  |
| --- |
| #include "cl\_5.h"  cl\_5::cl\_5(cl\_base\* parent, string name) : cl\_base(parent, name)  {} |

## **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\* parent, string name);  };  #endif |

## **Файл** **cl\_6.cpp**

Листинг 11 – cl\_6.cpp

|  |
| --- |
| #include "cl\_6.h"  cl\_6::cl\_6(cl\_base\* parent, string name) : cl\_base(parent, name)  {} |

## **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\* parent, string name);  };  #endif |

## **Файл** **cl\_application.cpp**

Листинг 13 – cl\_application.cpp

|  |
| --- |
| #include "cl\_application.h"  cl\_application::cl\_application(cl\_base\* parent) : cl\_base(parent)  {}  void cl\_application::build\_tree\_objects()  {  string objPath, childName, command;  int classNum;  cl\_base\* currentObj = this;  cin >> objPath;  SetName(objPath);  while(true)  {  cin.clear();  cin >> objPath;  if (objPath == "endtree")  {  break;  }  cl\_base\* parentObj = GetObjectByPath(objPath);  // Если указатель головной не найден  if (!parentObj)  {  // Выводим готовое дерево иерархии и завершаем выполнение программы  cout << "Object tree" << endl;  PrintObjects();  cout << "\nThe head object " << objPath << " is not found";  exit(1);  }  cin >> childName >> classNum;  // Если у головного объекта в подчиненных нет объекта с наименованием childName  if (!parentObj->GetChild(childName))  {  switch(classNum)  {  case 2:  {  new cl\_2(parentObj, childName);  break;  }  case 3:  {  new cl\_3(parentObj, childName);  break;  }  case 4:  {  new cl\_4(parentObj, childName);  break;  }  case 5:  {  new cl\_5(parentObj, childName);  break;  }  case 6:  {  new cl\_6(parentObj, childName);  break;  }  }  }  else  {  // Если обнаружен дубликат имени  cout << objPath << " Dubbing the names of subordinate objects" << endl;  }  }  // Вывод готового дерева  cout << "Object tree" << endl;  PrintObjects();  cout << endl;  while(true)  {  // Ввод команды для взаимодействия с объектами дерева  cin >> command;  if (command == "END")  {  break;  }  // Ввод пути объекта  cin >> objPath;  // Находим объект по введенному пути  cl\_base\* obj = currentObj->GetObjectByPath(objPath);  if (command == "SET")  {  if (obj)  {  // Устанавливаем новый текущий объект  currentObj = obj;  cout << "Object is set: " << obj->GetName() << endl;  }  else  {  cout << "The object was not found at the specified coordinate: " << objPath << endl;  }  }  else if (command == "FIND")  {  if (obj)  {  cout << objPath << " Object name: " << obj->GetName()  << endl;  }  else  {  cout << objPath << " Object is not found" << endl;  }  }  else if (command == "MOVE")  {  if (!obj)  {  cout << objPath << " Head object is not found" <<  endl;  }  // Если у текущего и найденного объектов разные родители  // И на ветке текущего содержится объект с наименованием объекта obj  else if (obj->GetParent() != currentObj->GetParent() && currentObj->SearchObjOnBranch(obj->GetName()))  {  cout << objPath << " Redefining the head object failed" << endl;  }  else if(obj->GetChild(currentObj->GetName()))  {  cout << objPath << " Dubbing the names of subordinate objects" << endl;  }  else if (!currentObj->ChangeHeadObj(obj))  {  cout << objPath << " Redefining the head object failed" << endl;  }  else  {  cout << "New head object: " << obj->GetName() << endl;  }  }  else if (command == "DELETE")  {  // Если объект найден  if(obj)  {  // Инициализируем временный указатель на головной объект текущего  cl\_base\* temp = obj->GetParent();  // Переменная для вывода полного пути удаленного объекта  string ans;  // Если объект удалился  if (currentObj->DeleteSubObj(obj->GetName()))  {  // Пока temp не равен nullptr  while(temp)  {  // Если головной объект текущего не равен nullptr  if (temp->GetParent())  ans += "/" + temp->GetName();  temp = temp->GetParent();  }  // Вывод информации о удалении объекта  cout << "The object " << ans + "/" + objPath << " has been deleted" << endl;  }  }  }  }  }  int cl\_application::exec\_app()  {  cout << "Current object hierarchy tree" << endl;  PrintObjects();  return 0;  } |

## **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\* parent);  void build\_tree\_objects();  int exec\_app();  };  #endif |

## **Файл** **cl\_base.cpp**

Листинг 15 – cl\_base.cpp

|  |
| --- |
| #include "cl\_base.h"  cl\_base::cl\_base(cl\_base\* parent, string name)  {  this->parent = parent;  this->name = name;  if (GetParent() != nullptr)  {  GetParent()->children.push\_back(this);  }  }  cl\_base::~cl\_base()  {  for (auto child : children)  {  delete child;  }  }  bool cl\_base::SetName(string newName)  {  // Если родитель для текущего найден и  // он не имеет подчиненнго с именем newName  if(GetParent() != nullptr && GetParent()->GetChild(newName) != nullptr)  {  return false;  }  name = newName;  return true;  }  string cl\_base::GetName() const  {  return name;  }  cl\_base\* cl\_base::GetParent() const  {  return parent;  }  cl\_base\* cl\_base::GetChild(string objName) const  {  // Проход по всем подчиненным объектам  for (auto child : children)  {  if (child->GetName() == objName)  {  return child;  }  }  return nullptr;  }  // Поиск количества объектов на ветке по имени  // через рекурсию  int cl\_base::ObjNameCount(string objName)  {  int count = 0;  if(GetName() == objName)  {  count++;  }  for (auto child : children)  {  count += child->ObjNameCount(objName);  }  return count;  }  // Проверка на уникальность в ветке по имени  cl\_base\* cl\_base::CheckingObjUniq(string objName)  {  if (ObjNameCount(objName) != 1)  {  return nullptr;  }  return SearchObjOnBranch(objName);  }  cl\_base\* cl\_base::SearchObjOnBranch(string objName)  {  // Если имя объекта равно параметру objname  if (GetName() == objName)  {  return this;  }  for (auto child : children)  {  cl\_base\* subChild = child->SearchObjOnBranch(objName);  // Если объект не равен nullptr  if (subChild)  {  return subChild;  }  }  return nullptr;  }  // Поиск по всему дереву от корневого  cl\_base\* cl\_base::SearchObjOnTree(string objName)  {  return GetRoot()->CheckingObjUniq(objName);  }  //Вывод  void cl\_base::PrintObjects(int spaces) const  {  cout << GetName();  if (!children.empty())  {  for (auto child : children)  {  cout << endl;  for (int i = 0; i < spaces; i++)  cout << " ";  child->PrintObjects(spaces+4);  }  }  }  void cl\_base::PrintObjectsStates(int spaces) const  {  cout << GetName();  cout << (GetObjectState() ? " is ready" : " is not ready");  if (!children.empty())  {  for (auto child : children)  {  cout << endl;  for (int i = 0; i < spaces; i++)  cout << " ";  child->PrintObjectsStates(spaces+4);  }  }  }  //Состояние объекта  void cl\_base::SetObjectState(bool state)  {  if (GetParent() && !GetParent()->GetObjectState())  {  this->state = false;  }  else  {  this->state = state;  }  if (!state)  {  for (auto child : children)  {  child->SetObjectState(state);  }  }  }  bool cl\_base::GetObjectState() const  {  return state;  }  cl\_base\* cl\_base::GetRoot()  {  cl\_base\* obj = this;  // Дохожим до корневого объекта  while(obj->GetParent())  {  obj = obj->GetParent();  }  return obj;  }  // Поиск объекта по пути  cl\_base\* cl\_base::GetObjectByPath(string path)  {  // Указатель на текущий  cl\_base\* currentObj = this;  string nextObjName;  if (path.substr(0,1) == "/")  {  if (path.substr(1,1) == "/")  {  return SearchObjOnTree(path.substr(2));  }  // Текущий равен корневому  currentObj = GetRoot();  path = path.substr(1);  }  else if (path.substr(0,1) == ".")  {  return path == "." ? currentObj : CheckingObjUniq(path.substr(1));  }  stringstream streamPath(path);  // Пока можно разбивать строку на подстроки по '/'  while(getline(streamPath, nextObjName, '/'))  {  currentObj = currentObj->GetChild(nextObjName);  // Если такой объект не найден  if (!currentObj)  {  return nullptr;  }  }  return currentObj;  }  // Смена головного объекта  bool cl\_base::ChangeHeadObj(cl\_base\* newHead)  {  // Если головной объект найден  if (GetParent())  {  for (auto i = (GetParent()->children).begin(); i != (GetParent()->children).end(); i++)  {  if (\*i == this)  {  // Удаляем у головного объекта текущего сам текущий  (GetParent()->children).erase(i);  break;  }  }  // Переопределяем головной объект  this->parent = newHead;  // Добавление объекта к новому головному  (GetParent()->children).push\_back(this);  return true;  }  return false;  }  // Удаление объекта  bool cl\_base::DeleteSubObj(string objName)  {  // Находим в подчиненных по имени  cl\_base\* subObj = GetChild(objName);  for (auto i = children.begin(); i != children.end(); i++)  {  if (\*i == subObj)  {  // Удаление объекта по его итератору в списке  children.erase(i);  delete subObj;  return true;  }  }  return false;  } |

## **Файл** **cl\_base.h**

Листинг 16 – cl\_base.h

|  |
| --- |
| #ifndef CL\_BASE\_H  #define CL\_BASE\_H  #include <iostream>  #include <string>  #include <sstream>  #include <vector>  using namespace std;  class cl\_base  {  string name;  cl\_base\* parent;  bool state = false;  vector<cl\_base\*> children;  public:  cl\_base(cl\_base\* parent, string name = "Base\_object");  ~cl\_base();  // Установка наименования для текущего объекта  bool SetName(string newName);  // Возврат значения наименования объекта  string GetName() const;  // Возврат указателя на головной объект  cl\_base\* GetParent() const;  // Возврат указателя на подчиненный объект  cl\_base\* GetChild(string objName) const;  // Проверка на уникальность  int ObjNameCount(string objName);  // Проверка объекта на уникальность на ветке  cl\_base\* CheckingObjUniq(string objName);  // Поиск объекта на ветке по наименованию  cl\_base\* SearchObjOnBranch(string objName);  // Поиск объекта на дереве по наименованию  cl\_base\* SearchObjOnTree(string objName);  // Вывод дерева иерархии объектов  void PrintObjects(int spaces = 4) const;  // Методы для состояния объекта - КВ2  void PrintObjectsStates(int spaces = 4) const;  // Установка состояния объекта  void SetObjectState(bool state);  // Возврат состояния объекта  bool GetObjectState() const;  // Возврат указателя на корневой объект иерархии  cl\_base\* GetRoot();  // Возврат указателя на объект иерархии по передаваемому пути  cl\_base\* GetObjectByPath(string path);  // Переопределение головного объекта для текущего в дереве иерархии  bool ChangeHeadObj(cl\_base\* newHead);  // Удаление объекта у текущего в дереве иерархии  bool DeleteSubObj(string objName);  };  #endif |

## **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 10.

Таблица 10 – 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  DELETE object\_2  END | Object tree  rootela  object\_1  object\_7  object\_2  object\_4  object\_7  object\_5  object\_3  object\_3  The object /object\_2 has been deleted  Current object hierarchy tree  rootela  object\_1  object\_7  object\_3 | Object tree  rootela  object\_1  object\_7  object\_2  object\_4  object\_7  object\_5  object\_3  object\_3  The object /object\_2 has been deleted  Current object hierarchy tree  rootela  object\_1  object\_7  object\_3 |
| 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 |
| 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  FIND //object\_7  SET /object\_2  FIND //object\_7  FIND object\_4/object\_7  FIND /././.  FIND ././.  FIND ///  FIND .object\_7  FIND object\_4/object\_7  MOVE .object\_7  SET object\_4/object\_7  MOVE //object\_1  MOVE /object\_3  SET /object\_2  DELETE object\_7  DELETE object\_5  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\_7 Object is not found  Object is set: object\_2  //object\_7 Object is not found  object\_4/object\_7 Object name: object\_7  /././. Object is not found  ././. Object is not found  /// Object is not found  .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  Object is set: object\_2  The object /object\_2/object\_5 has been deleted  Current object hierarchy tree  rootela  object\_1  object\_7  object\_2  object\_4  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\_7 Object is not found  Object is set: object\_2  //object\_7 Object is not found  object\_4/object\_7 Object name: object\_7  /././. Object is not found  ././. Object is not found  /// Object is not found  .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  Object is set: object\_2  The object /object\_2/object\_5 has been deleted  Current object hierarchy tree  rootela  object\_1  object\_7  object\_2  object\_4  object\_3  object\_3  object\_7 |
| r  / a 2  / b 3  /a b 3  /a/b c 4  /a/b/c a 2  /a/b/c/a a 2  /a/b/c/a b 3  endtree  SET /a  MOVE /b  END | Object tree  r  a  b  c  a  a  b  b  Object is set: a  New head object: b  Current object hierarchy tree  r  b  a  b  c  a  a  b | Object tree  r  a  b  c  a  a  b  b  Object is set: a  New head object: b  Current object hierarchy tree  r  b  a  b  c  a  a  b |

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe\_posobie\_dlya\_laboratornyh\_rabot\_3.pdf (дата обращения 05.05.2021).

3. 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).

4. Шилдт Г. С++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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).