

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 8](#_Toc105803382)

[1  ПОСТАНОВКА ЗАДАЧИ 9](#_Toc105803383)

[2  МЕТОД РЕШЕНИЯ 14](#_Toc105803386)

[3  ОПИСАНИЕ АЛГОРИТМОВ 18](#_Toc105803387)

[4  БЛОК-СХЕМЫ АЛГОРИТМОВ 36](#_Toc105803414)

[ЗАКЛЮЧЕНИЕ 57](#_Toc105803415)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 58](#_Toc105803416)

[Приложение А 59](#_Toc105803417)

[Приложение Б 68](#_Toc105803428)

# ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД). Все этапы решения задач курсовой работы фиксированы, соответствуют методике разработки объектно- ориентированных программ [1-2] и требованиям, приведенным в методическом пособии для проведения практических заданий контрольных и курсовых работ по дисциплине "Объектно-ориентированное-программирование" [3-4].

Цель работы: повышение практических навыков в области проектирование и реализации задач на основных принципах объектно-ориентированного программирования.

Задача: разработать инженерный арифметический калькулятор. Реализовать алгоритм выполнения операций над целыми числами.

Инженерный арифметический калькулятор упрощает процесс выполнения операций над числами. Также калькулятор позволяет представить число в различных системах счисления.

Инженерный арифметический калькулятор используется студентами в решении более сложных задач, где калькулятор выполняет второстепенную роль вычисления чисел. Кроме того, он исключает вероятность вычислительной ошибки.

# 1 ПОСТАНОВКА ЗАДАЧИ

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

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задается в следующем виде:

/                  - корневой объект;

//«имя объекта»    - поиск объекта по уникальному имени от корневого (для однозначности уникальность требуется в рамках дерева);

.                  - текущий объект;

«имя объекта 1»[/«имя объекта 2»] . . .    - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;

/«имя объекта 1»[/«имя объекта 2»] . . .   - абсолютная координата от корневого объекта.

Примеры координат:

/

//ob\_3

.

ob\_2/ob\_3

ob\_2

/ob\_1/ob\_2/ob\_3

Если координата пустая строка или объект не найден, то вернуть нулевой указатель.

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

Состав и иерархия объектов строиться посредством ввода исходных данных.  Ввод организован как в версии № 2 курсовой работы.

Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта соблюдены.

Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов).

Система отрабатывает следующие команды:

SET «координата»  – устанавливает текущий объект;

FIND «координата» – находит объект относительно текущего;

END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим.

При вводе данных в названии команд ошибок нет. Условия уникальности имен объектов для однозначной отработки соответствующих команд соблюдены.

## 1.1 Описание входных данных

Состав и иерархия объектов строится посредством ввода исходных данных.  Ввод организован как в версии № 2 курсовой работы.

Единственное различие, в строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

SET «координата»  - установить текущий объект;

FIND «координата» - найти объект относительно текущего;

END   – завершить функционирование системы (выполнение программы).

Команды SET и FIND вводятся произвольное число раз. Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов.

root

/ object\_1 3

/ object\_2 2

/object\_2 object\_4 3

/object\_2 object\_5 4

/ object\_3 3

/object\_2 object\_3 6

/object\_1 object\_7 5

/object\_2/object\_4 object\_7 3

endtree

FIND object\_2/object\_4

SET /object\_2

FIND //object\_5

FIND /object\_15

FIND .

FIND object\_4/object\_7

END

## 1.2 Описание выходных данных

Первая строка:

Object tree

Со второй строки вывести иерархию построенного дерева как в курсовой работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение:

The head object «координата головного объекта» is not found и прекратить работу программы.

Если дерево построено, то далее построчно:

для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

Object is not found: «имя текущего объекта» «искомая координата объекта»

для команд FIND вывести:

«искомая координата объекта»     Object name: «наименование объекта»

Если объект не найден, то:

«искомая координата объекта»     Object is not found

Пример вывода иерархии дерева объектов.

Object tree

root

    object\_1

        object\_7

    object\_2

        object\_4

            object\_7

        object\_5

        object\_3

    object\_3

object\_2/object\_4     Object name: object\_4

Object is set: object\_2

//object\_5     Object name: object\_5

/object\_15     Object is not found

.     Object name: object\_2

object\_4/object\_7     Object name: object\_7

# 2 МЕТОД РЕШЕНИЯ

Для решения поставленной задачи требуется использовать:

1. Объекты стандартных потоков ввода и вывода. Используются для ввода с клавиатуры и вывода на экран
2. Объект ob\_cl\_application класса cl\_application
3. Условный оператор if
4. Операторы циклов for, while
5. Объекты классов Base, cl\_application, class1, class2, class3, class4, class5, class6
6. Вектор пар для хранения данных

**Класс Base:**

* свойства / поля:
  + поле – хранение названия текущего объекта:
    - наименование – name;
    - тип - string;
    - модификатор доступа – private.
  + поле – хранение указателя на родителя для текущего объекта:
    - наименование – \*parent;
    - тип – Base;
    - модификатор доступа – private.
  + поле – массив указателей на дочерние объекты текущего объекта:
    - наименование – children;
    - тип - vector;
    - модификатор доступа – private.
  + поле - хранение состояния объекта:
    - наименование – x;
    - тип – int;
    - модификатор доступа – private.
* функционал:
  + конструктор Base - параметризированный конструктор;
  + метод set\_name - определение имени объекта;
  + метод get\_name - получение имени объекта;
  + метод set\_root – определение родителя;
  + метод get\_root – получение указателя на родителя;
  + метод print\_tree - вывод на экран веток дерева иерархии;
  + метод find\_object - поиск объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr);
  + метод print\_ready\_tree - вывод дерева иерархии объектов и отметок их готовности;
  + метод set\_ready - установка готовности объекта;
  + метод get\_ready - возврат значения готовности объекта;
  + метод get\_children – возврат вектора указателей на детей текущего объекта;
  + метод find\_coordinate - возврат указателя на объект согласно пути, заданному пользователем;

**Класс cl\_application:**

* + функционал:
  + конструктор cl\_application - параметризированный конструктор;
  + метод build\_tree\_objects - построение дерева иерархии объектов;
  + метод exec\_app - запуск приложения.

**Класс class1 - наследуется публично от класса Base:**

* + функционал:
  + конструктор class1 – параметризированный конструктор.

**Класс class2 - наследуется публично от класса Base:**

* + функционал:
  + конструктор class2 – параметризированный конструктор.

**Класс class3 - наследуется публично от класса Base:**

* + функционал:
  + конструктор **class3** – параметризированный конструктор.

**Класс class4 - наследуется публично от класса Base:**

* + функционал:
  + конструктор **class4** – параметризированный конструктор.

**Класс class5 - наследуется публично от класса Base:**

* + функционал:
  + конструктор **class5** – параметризированный конструктор.

**Класс class6 - наследуется публично от класса Base:**

* + функционал:
  + конструктор class6 – параметризированный конструктор.

Таблица 1 – Иерархия наследования объектов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | **Имя класса** | **Классы-наследники** | **Модификатор доступа при наследовании** | **Описание** | **Номер** | **Комментарии** |
| 1 | Base | cl\_application | public |  | 2 |  |
| class1 | public |  | 3 |  |
| class2 | public |  | 4 |  |
| class3 | public |  | 5 |  |
| class4 | public |  | 6 |  |
| class5 | public |  | 7 |  |

*Продолжение Таблицы 1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 |  | class6 | public |  | 8 |  |
| 2 | cl\_application |  |  | Содержит метод построения дерева иерархии и метод запуска приложения |  |  |
| 3 | class1 |  |  | Класс с параметризированным конструктором первого объекта |  |  |
| 4 | class2 |  |  | Класс с параметризированным конструктором первого объекта |  |  |
| 5 | class3 |  |  | Класс с параметризированным конструктором первого объекта |  |  |
| 6 | class4 |  |  | Класс с параметризированным конструктором первого объекта |  |  |
| 7 | class5 |  |  | Класс с параметризированным конструктором первого объекта |  |  |
| 8 | class6 |  |  | Класс с параметризированным конструктором первого объекта |  |  |

# 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 функционал программы.

Параметры: отсутствуют.

Возвращаемое значение: результат алгоритма.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Создание объекта ob\_cl\_application класса cl\_application | 2 |
| 2 |  | Вызов метода build\_tree\_objects класса cl\_application | 3 |
| 3 |  | Возврат вызова метода exec\_app класса cl\_application | ∅ |

## Алгоритм конструктора класса Base

Функционал: параметризированный конструктор.

Параметры: имя текущего объекта и указатель на родителя.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса Base

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Объявление строковых переменных par, kid | 2 |
| *Продолжение таблицы 3* | | | |
| 2 |  | Объявление целочесленной n = 0 | 3 |
| № | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Присвоение name к именитекущего объекта | 2 |
| 2 |  | Присвоение parent кродителю текущегообъекта | 3 |
| 3 | Указатель на родителя существует | Добавление текущегообъекта в массив объектов данного родителя | ∅ |
|  |  | ∅ |

## Алгоритм метода set\_name класса Base

Функционал: определение имени объекта.

Параметры: string name – имя объекта.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set\_name класса Base

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Присвоение переменной name имени текущего объекта | ∅ |

## Алгоритм метода set\_root класса Base

Функционал: определение родителя.

Параметры: new\_parent - указатель на родителя.

Возвращаемое значение: отсутствует.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм метода set\_root класса Base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Указатель на родителя |  | 2 |
|  |  | 5 |
| 2 | i<размер children |  | 3 |
|  |  | 5 |
| 3 | Значение в массиве==значение текущего объекта | Удаление текущего значения из children | 4 |
|  |  | 2 |
| 4 |  | i+1 | 2 |
| 5 |  | Присвоение переменной parent значения переменной new\_parent | 6 |
| 6 | Указатель на нового родителя существует | Добавление текущего объекта в массив объектов нового родителя | ∅ |
|  |  | ∅ |

## Алгоритм метода get\_name класса Base

Функционал: получение имени объекта.

Параметры: отсутствуют.

Возвращаемое значение: строка – имя объекта

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм метода get\_name класса Base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Возврат значения переменной name | ∅ |

## Алгоритм метода get\_root класса Base

Функционал: получение указателя на родителя.

Параметры: отсутствуют.

Модификатор доступа: public

Возвращаемое значение: указатель на родителя текущего объекта.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода get\_root класса Base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Возврат значения переменной parent | ∅ |

## Алгоритм метода print\_tree класса Base

Функционал: вывод на экран веток дерева иерархии.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода print\_tree класса Base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Переход на новую строку,вывод значения параметра space и вызов метода get\_name() | 2 |
| 2 | Перебор элементов в массиве children | Вызов метода print\_tree(space + " ") для child | 2 |
|  |  | ∅ |

## Алгоритм конструктора класса cl\_application

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl\_application

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Наследование от конструктора класса Base | ∅ |

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

Функционал: построение дерева иерархии объектов.

Параметры: отсутствуют.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build\_tree\_objects класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление строковых переменных parent\_name и name | 2 |
| 2 |  | Объявление указателей parent и child класса Base | 3 |
| 3 |  | Считывание с клавиатуры значения переменной parent\_name | 4 |
| 4 |  | Вызов метода set\_name с параметром parent\_name | 5 |
| 5 |  | Объявление целочисленной переменной num | 6 |
| 6 | true | Считывание с клавиатуры значения переменной parent\_name | 7 |
|  |  | ∅ |
| 7 | parent\_name=="endtree" |  | ∅ |
|  |  | 8 |
| 8 |  | Считывание с клавиатуры значений переменных name и num | 9 |

Продолжение Таблицы 10

|  |  |  |  |
| --- | --- | --- | --- |
| 9 |  | Присвоение переменной parent метода  find\_object(parent\_name) | 10 |
| 10 | num == 2 | child = new class1(name,parent) | 6 |
|  |  |  | 11 |
| 11 | num == 3 | child = new class2(name,parent) | 6 |
|  |  |  | 12 |
| 12 | num == 4 | child = new class3(name,parent) | 6 |
|  |  |  | 13 |
| 13 | num == 5 | child = new class4(name,parent) | 6 |
|  |  |  | 14 |
| 14 | num == 6 | child = new class5(name,parent) | 6 |
|  |  |  | 15 |

## Алгоритм метода exec\_app класса cl\_application

Функционал: запуск приложения.

Параметры: отсутствуют.

Возвращаемое значение: Целочисленное значение - код возврата.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec\_app класса cl\_application

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Вывод сообщения "Object tree" на экран | 2 |
| 2 |  | Вызов метода print\_tree("") | 3 |
| 3 |  |  | 4 |
| 4 |  |  | 5 |
| *Продолжение таблицы 11* | | | |
| 5 | Считывание с клавиатуры object\_name и status | Инициализация указателя find = find\_object(object\_name) | 6 |
|  |  | 7 |
| 6 |  | Вызов метода set\_ready(status) для find | 5 |
| 7 |  | Переход на новую строку и вывод сообщения "The tree of objects and their readiness" на экран | 8 |
| 8 |  | Вызов метода print\_ready\_tree("") | 9 |
| 9 |  | Возврат 0 | ∅ |

## Алгоритм конструктора класса class1

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base\* parent - указатель на родителя.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса class1

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name к имени текущего объекта | 2 |
| 2 |  | Присвоение parent к родителю текущегообъекта | 3 |
| 3 | Указатель на родителя существует | Добавление текщего объекта в массив объектов данного родителя | ∅ |
|  |  | ∅ |

## Алгоритм конструктора класса class2

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base\* parent - указатель на родителя.

Алгоритм метода представлен в таблице 13.

Таблица 3 – Алгоритм конструктора класса class2

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name к имени текущего объекта | 2 |
| 2 |  | Присвоение parent к родителю текущегообъекта | 3 |
| 3 | Указатель на родителя существует | Добавление текщего объекта в массив объектов данного родителя | ∅ |
|  |  | ∅ |

## Алгоритм конструктора класса class3

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base\* parent - указатель на родителя.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса class3

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name к имени текущего объекта | 2 |
| 2 |  | Присвоение parent к родителю текущегообъекта | 3 |
| 3 | Указатель на родителя существует | Добавление текщего объекта в массив объектов данного родителя | ∅ |
|  |  | ∅ |

## Алгоритм конструктора класса class4

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base\* parent - указатель на родителя.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса class4

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name к имени текущего объекта | 2 |
| 2 |  | Присвоение parent к родителю текущегообъекта | 3 |
| 3 | Указатель на родителя существует | Добавление текщего объекта в массив объектов данного родителя | ∅ |
|  |  | ∅ |

## Алгоритм конструктора класса class5

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base\* parent - указатель на родителя.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм конструктора класса class5

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name к имени текущего объекта | 2 |
| 2 |  | Присвоение parent к родителю текущегообъекта | 3 |
| 3 | Указатель на родителя существует | Добавление текщего объекта в массив объектов данного родителя | ∅ |
|  |  | ∅ |

## Алгоритм конструктора класса class6

Функционал: параметризированный конструктор.

Параметры: string name - имя объекта, Base\* parent - указатель на родителя.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм конструктора класса class6

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name к имени текущего объекта | 2 |
| 2 |  | Присвоение parent к родителю текущегообъекта | 3 |
| 3 | Указатель на родителя существует | Добавление текщего объекта в массив объектов данного родителя | ∅ |
|  |  |  | ∅ |

## Алгоритм метода find\_object класса Base

Функционал: поиск объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr).

Параметры: string name - имя объекта.

Возвращаемое значение: указатель на объект.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода find\_object класса Base

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 | Имя текущего объекта == name | Возврат указателя на объект | ∅ |
|  |  | 2 |
| 2 | перебор элементов массива children | Инициализация указателяres с вызовом метода find\_object для элементовмассива children | 3 |
|  |  | 4 |
| 3 | res существует | Возврат res | 2 |
|  |  | 2 |
| 4 |  | Возврат нулевого указателя | ∅ |

## Алгоритм метода print\_ready\_tree класса Base

Функционал: вывод дерева иерархии объектов и отметок их готовности.

Параметры: string space.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода print\_ready\_tree класса Base

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Переход на новую строку,вывод значения параметраspace и вызов метода get\_name() | 2 |
| 2 | x == false | Вывод сообщения " is not ready" на экран | 4 |
|  |  | 3 |
| 3 |  | Вывод сообщения " is ready" на экран | 4 |
| 4 | перебор элементов в массиве children | Вызов метода print\_ready\_tree(space + " ") для  child | 4 |
|  |  | ∅ |

## Алгоритм метода set\_ready класса Base

Функционал: установка готовности объекта.

Параметры: отсутствуют.

Возвращаемое значение: int x - готовность текущего объекта.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода findByRelPath класса Root

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 | !x |  | 2 |
|  |  | 6 |

*Продолжение Таблицы 21*

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | parent существует и готовность parent== 0 | Возврат true | ∅ |
|  |  | 3 |
| 3 | перебор элементов в массиве children | Для всех элементов вмассиве children готовность == 0 | 3 |
|  |  | 4 |
| 4 |  | x == 0 для текущего объекта | 5 |
| 5 |  | Возврат true | ∅ |
| 6 | parent существует и готовность parent == 0 | Возврат false | ∅ |
|  |  | 7 |
| 7 |  | Присвоение x текущего объекта к x | 8 |
| 8 |  | Возврат true | ∅ |

## 3.20 Алгоритм метода get\_ready класса Base

Функционал: возврат значения готовности объекта.

Параметры: отсутствуют.

Возвращаемое значение: int x - готовность текущего объекта.

Алгоритм функции представлен в таблице 22.

Таблица 22 – Алгоритм метода get\_ready класса Base

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Возврат значения готовности  текущего объекта | ∅ |

## 3.22 Алгоритм get\_children класса Base

Функционал: возврат вектора указателей на детей текущего объекта.

Параметры: отсутствуют.

Возвращаемое значение: вектор указателей.

Алгоритм конструктора представлен в таблице 23.

Таблица 23 – Алгоритм конструктора класса Child2

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Возврат вектора children для текущего объекта | ∅ |

## 3.23 Алгоритм метода find\_coordinate класса Base

Функционал: Параметризированный конструктор.

Параметры: Строковая \_name - имя объекта, указатель на родительский объект.

Алгоритм конструктора представлен в таблице 24.

Таблица 24 – Алгоритм конструктора класса Child3

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name значения \_name из параметров | 2 |
| 2 | ptrToPar == nullptr | Вызов метода setParent с параметром объектом rt класса Root | 3 |
|  | присвоение полю parent текущего объекта значения ptrToPar | 4 |
| 3 |  | Добавление текущего объекта в список kids родительского объекта | ∅ |
| 4 |  | Добавление ptrToPar в список kids текущего объекта | ∅ |

## 3.24 Алгоритм конструктора класса Child4

Функционал: Параметризированный конструктор.

Параметры: Строковая \_name - имя объекта, указатель на родительский объект.

Алгоритм конструктора представлен в таблице 25.

Таблица 25 – Алгоритм конструктора класса Child4

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name значения \_name из параметров | 2 |
| 2 | ptrToPar == nullptr | Вызов метода setParent с параметром объектом rt класса Root | 3 |
|  | присвоение полю parent текущего объекта значения ptrToPar | 4 |
| 3 |  | Добавление текущего объекта в список kids родительского объекта | ∅ |
| 4 |  | Добавление ptrToPar в список kids текущего объекта | ∅ |

## 3.25 Алгоритм конструктора класса Child5

Функционал: Параметризированный конструктор.

Параметры: Строковая \_name - имя объекта, указатель на родительский объект.

Алгоритм конструктора представлен в таблице 26.

Таблица 26 – Алгоритм конструктора класса Child5

|  |  |  |  |
| --- | --- | --- | --- |
| № | Предикат | Действия | № перехода |
| 1 |  | Присвоение name значения \_name из параметров | 2 |
| 2 | ptrToPar == nullptr | Вызов метода setParent с параметром объектом rt класса Root | 3 |
|  | присвоение полю parent текущего объекта значения ptrToPar | 4 |
| 3 |  | Добавление текущего объекта в список kids родительского объекта | ∅ |
| 4 |  | Добавление ptrToPar в список kids текущего объекта | ∅ |

## 3.26 Алгоритм конструктора класса Child6

Функционал: Параметризированный конструктор.

Параметры: Строковая \_name - имя объекта, указатель на родительский объект.

Алгоритм конструктора представлен в таблице 27.

Таблица 27 – Алгоритм конструктора класса Child6

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение name значения \_name из параметров | 2 |
| 2 | ptrToPar == nullptr | Вызов метода setParent с параметром объектом rt класса Root | 3 |
|  | присвоение полю parent текущего объекта значения ptrToPar | 4 |
| 3 |  | Добавление текущего объекта в список kids родительского объекта | ∅ |
| 4 |  | Добавление ptrToPar в список kids текущего объекта | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-21.

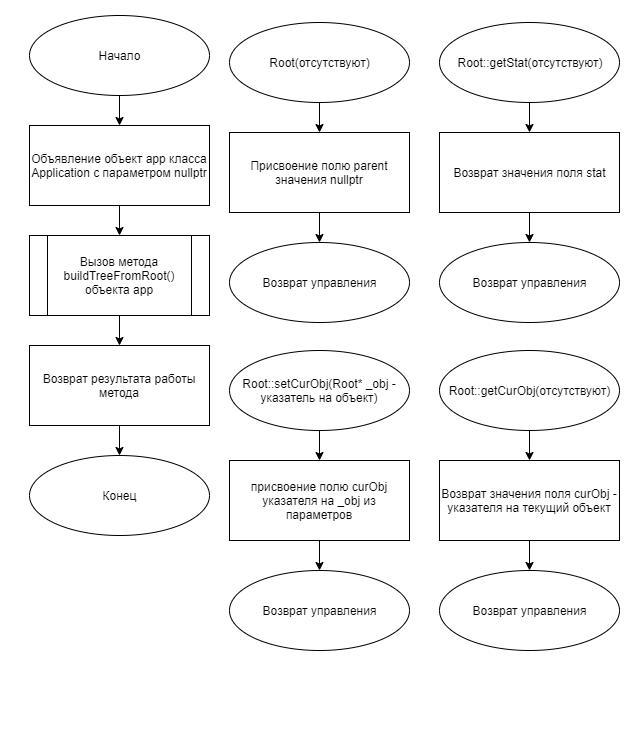


Рисунок 1 – Блок-схема алгоритма

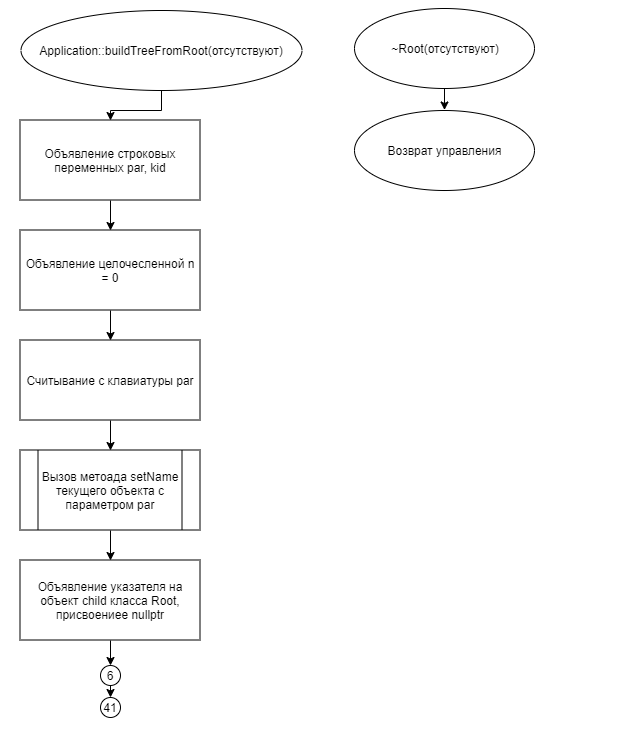


Рисунок 2 – Блок-схема алгоритма

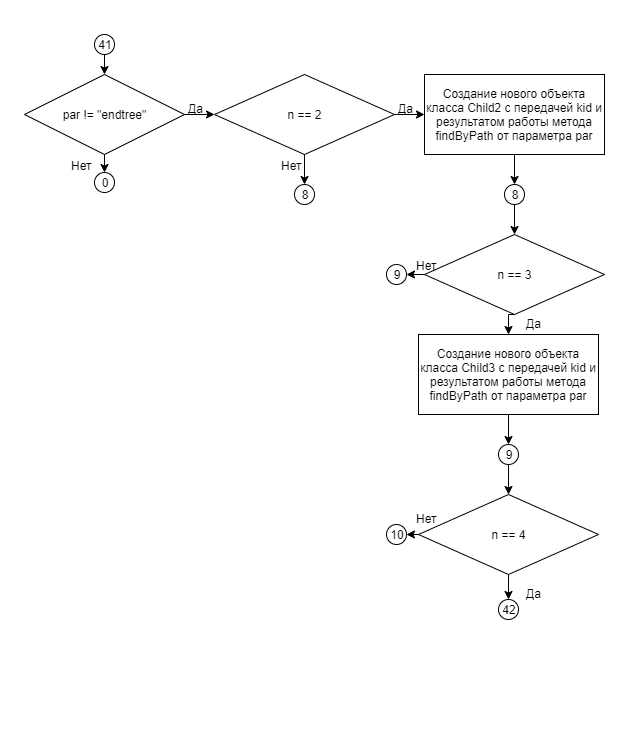


Рисунок 3 – Блок-схема алгоритма

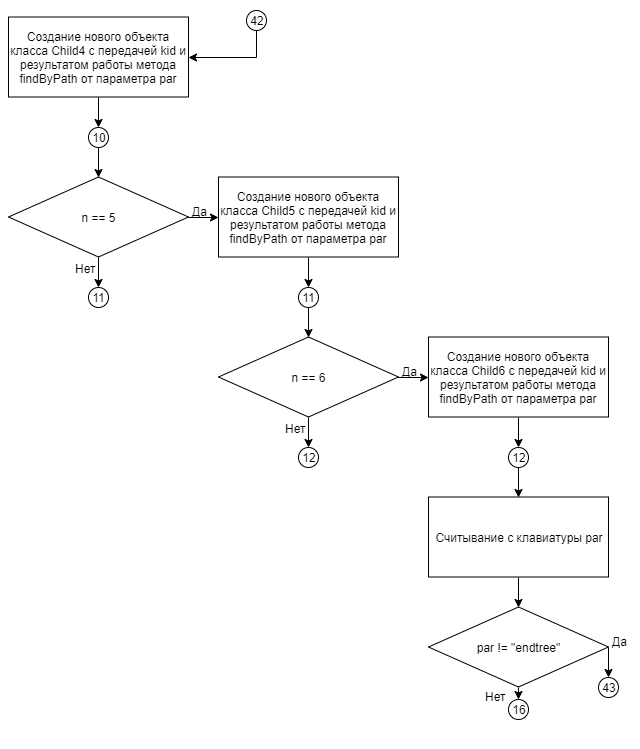


Рисунок 4 – Блок-схема алгоритма

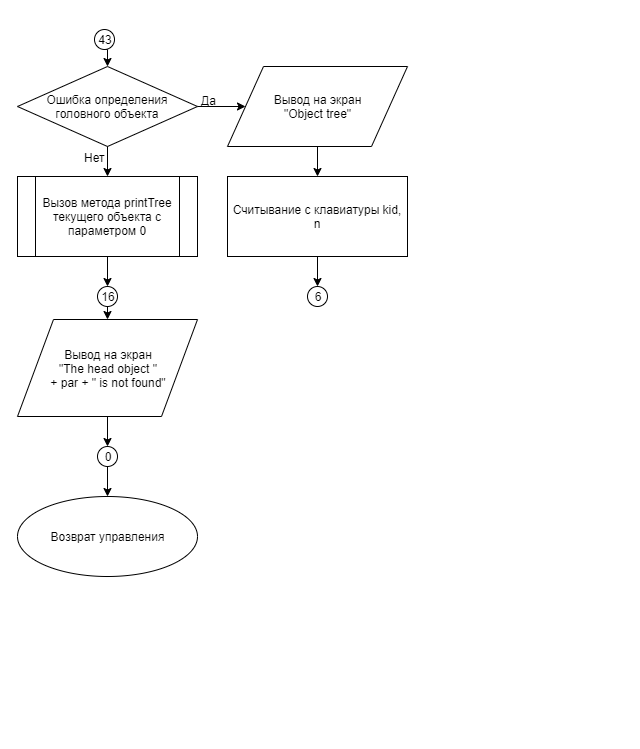


Рисунок 5 – Блок-схема алгоритма

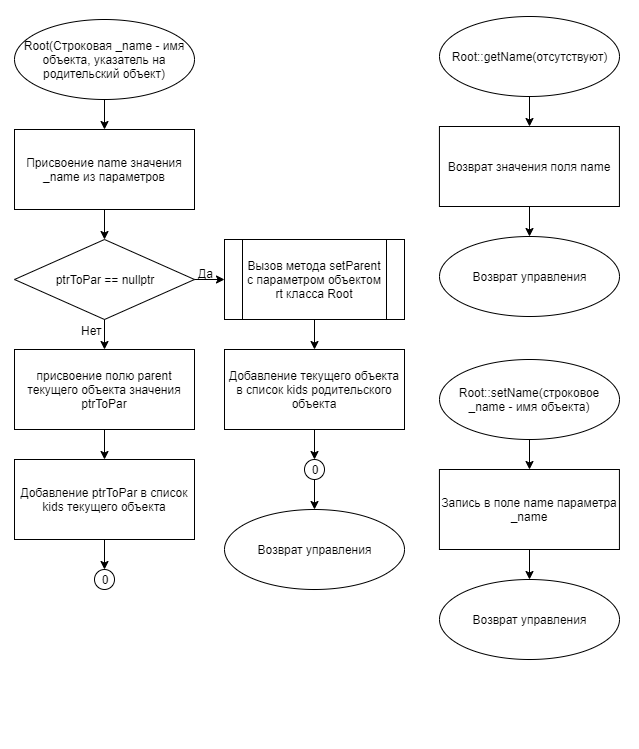


Рисунок 6 – Блок-схема алгоритма

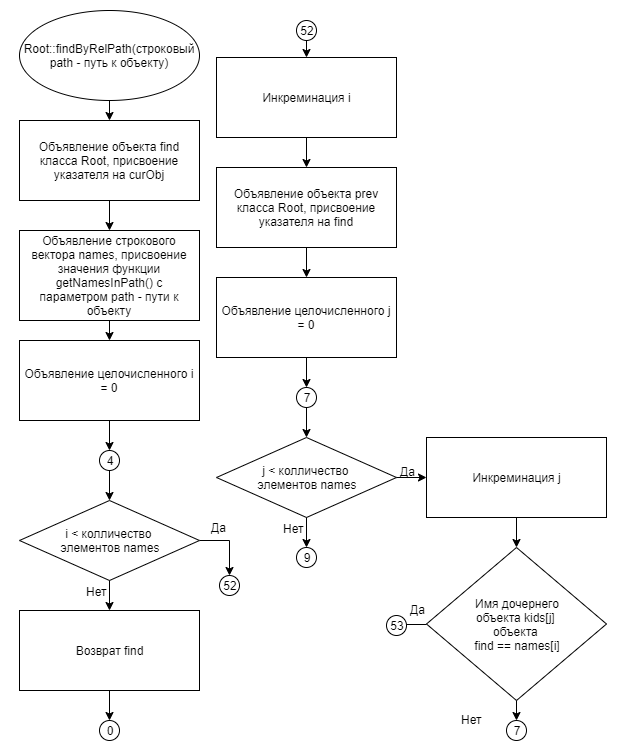


Рисунок 7 – Блок-схема алгоритма

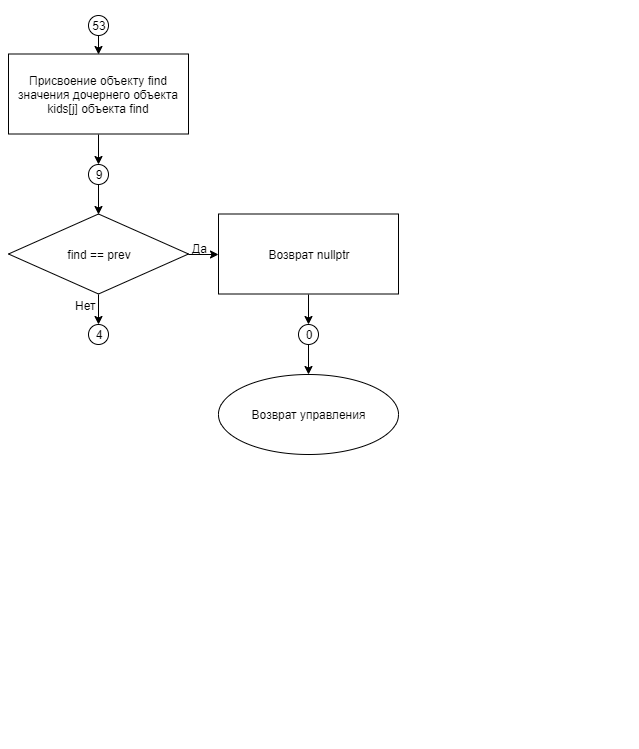


Рисунок 8 – Блок-схема алгоритма

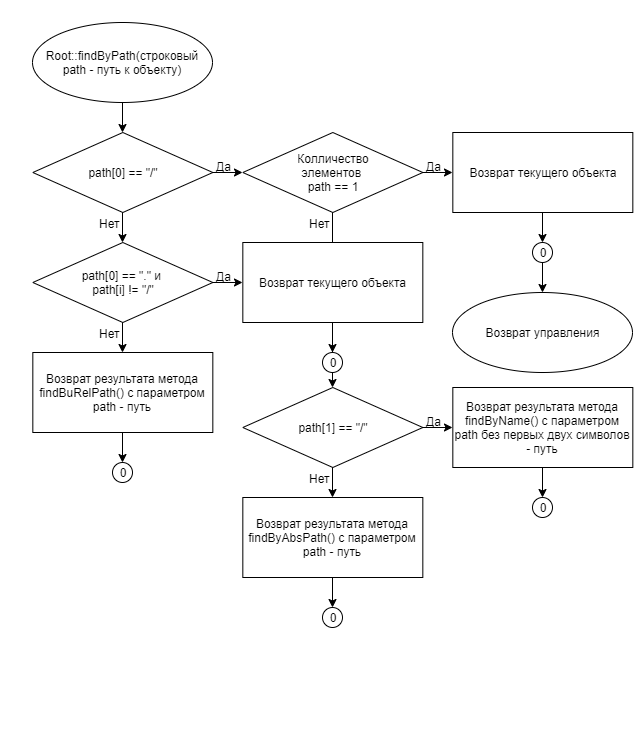


Рисунок 9 – Блок-схема алгоритма

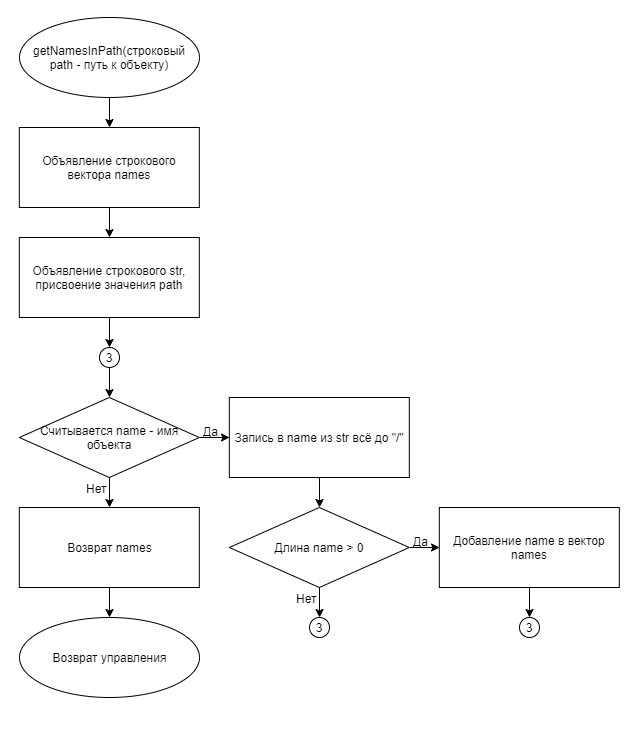


Рисунок 10 – Блок-схема алгоритма

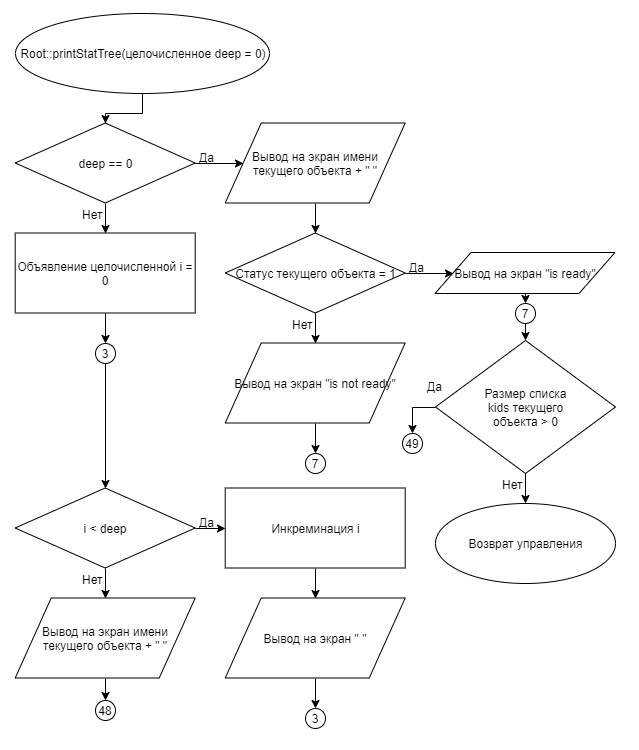


Рисунок 11 – Блок-схема алгоритма

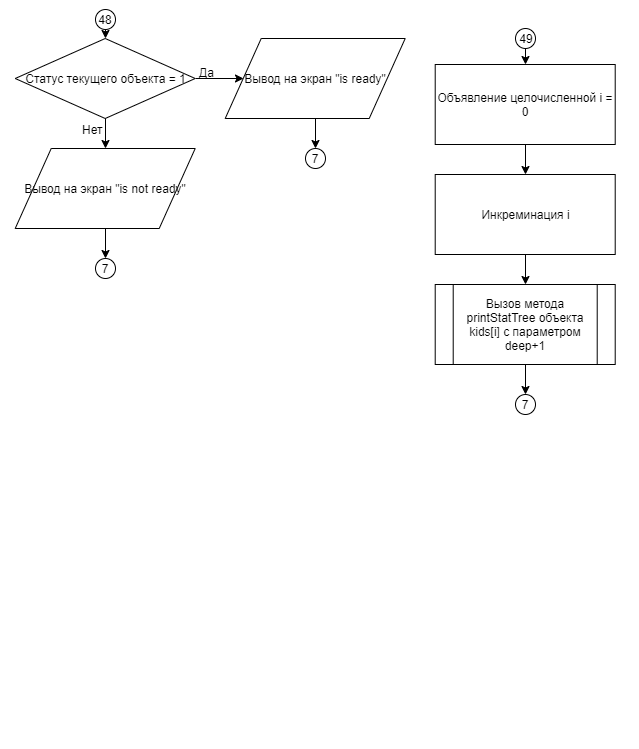


Рисунок 12 – Блок-схема алгоритма

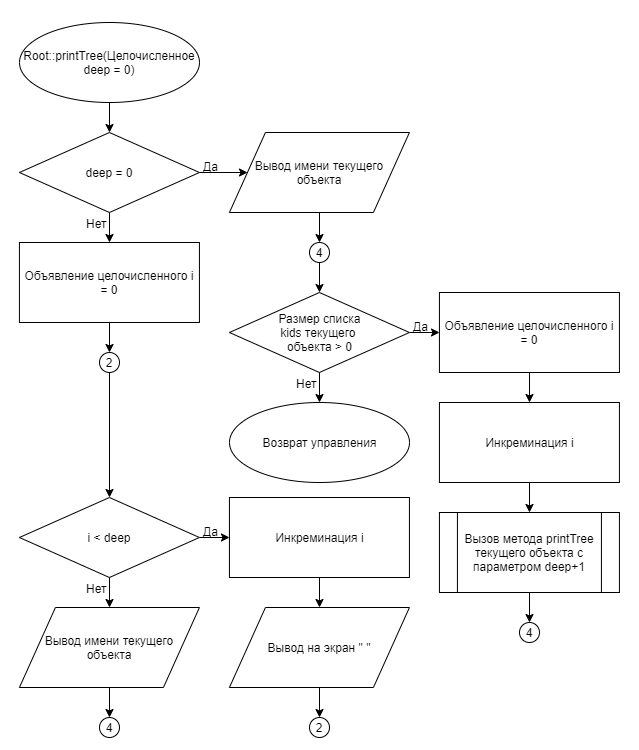


Рисунок 13 – Блок-схема алгоритма

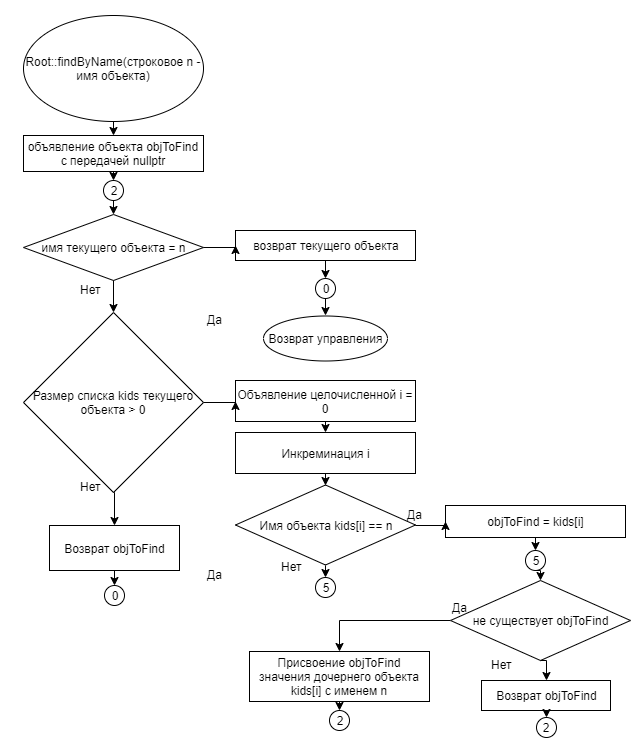


Рисунок 14 – Блок-схема алгоритма

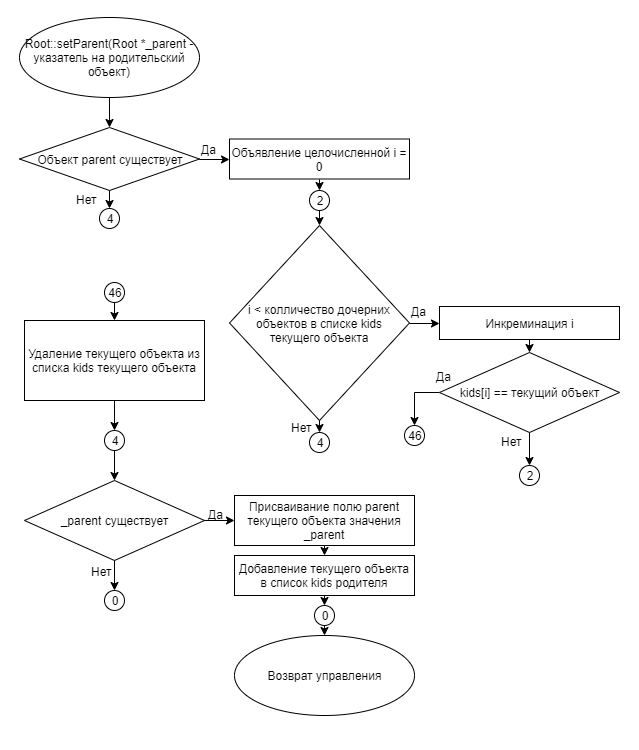


Рисунок 15 – Блок-схема алгоритма

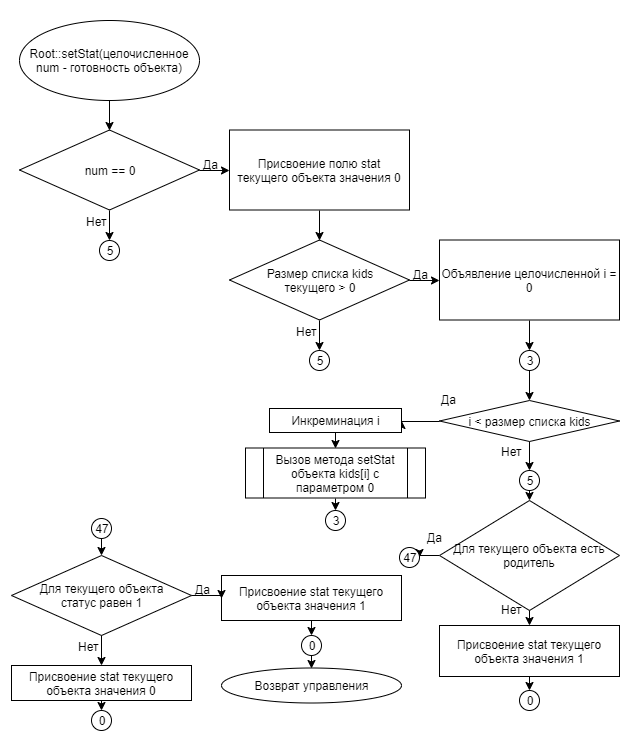


Рисунок 16 – Блок-схема алгоритма

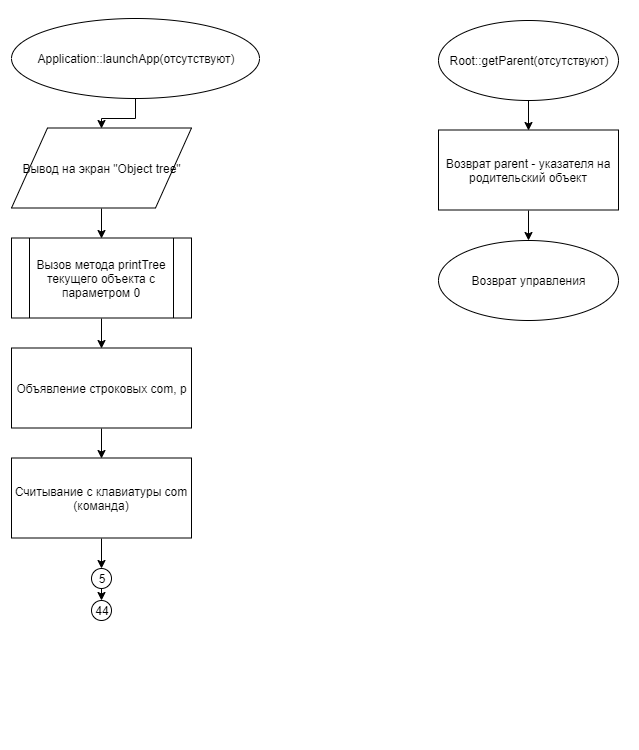


Рисунок 17 – Блок-схема алгоритма

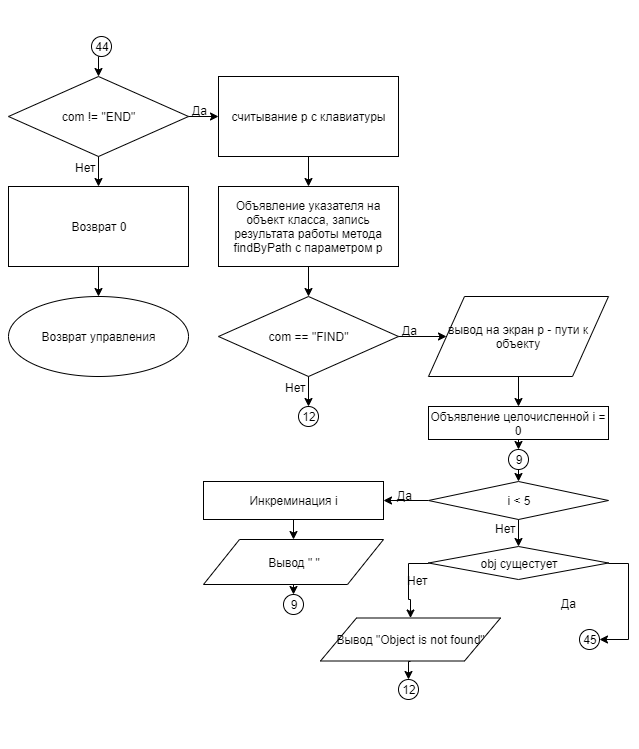
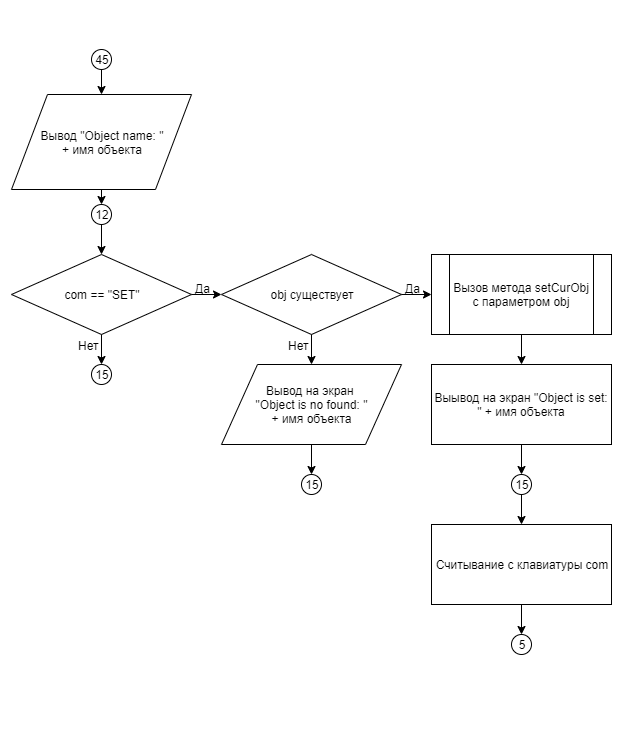


Рисунок 18 – Блок-схема алгоритма

Рисунок 19 – Блок-схема алгоритма

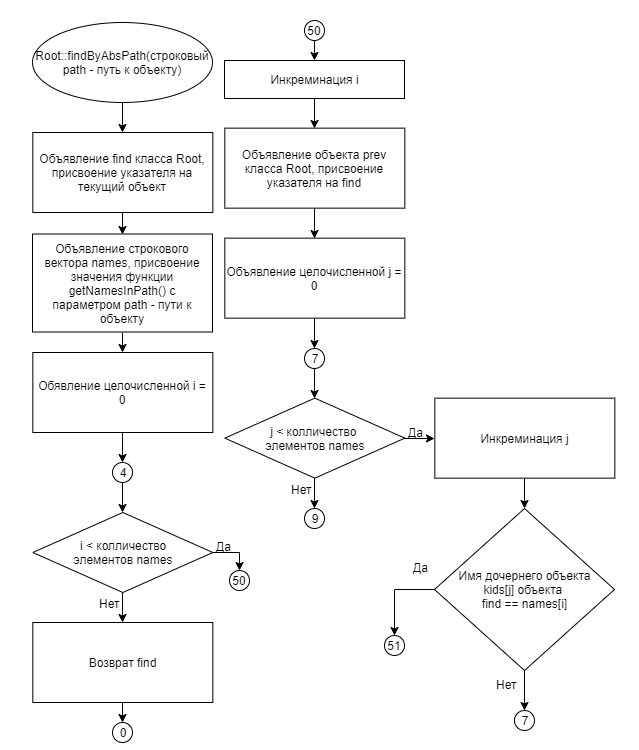


Рисунок 20 – Блок-схема алгоритма

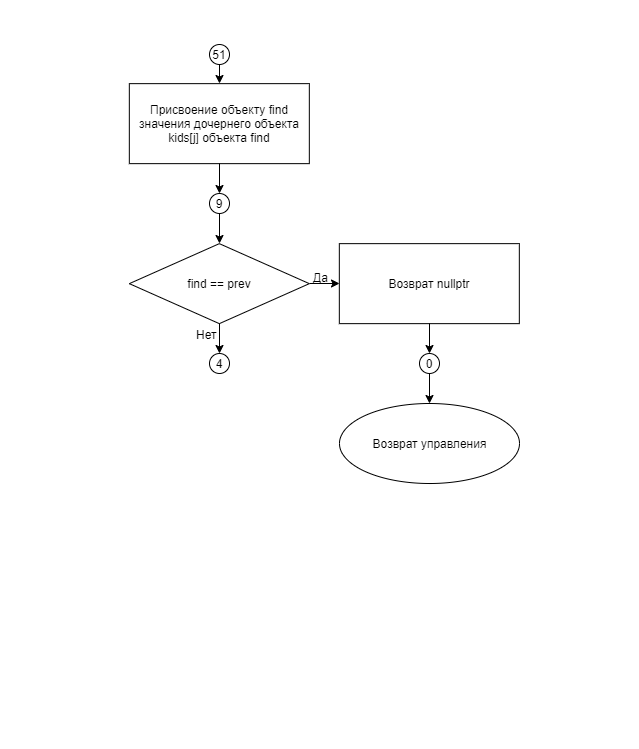


Рисунок 21 – Блок-схема алгоритма

# ЗАКЛЮЧЕНИЕ

В процессе выполнения работы было разработано и реализовано решение поставленной задачи. Описана четкая методология решения, составлена алгоритмическая и графическая составляющая задачи, сформирован исходный код задачи с логическим разделением на подзадачи и распределением этих подзадач по соответствующим файлам. Были проведены отладка и автоматизированное тестирование кода.

По итогам выполнения данной работы были повышены практические навыки в области объектно-ориентированного программирования, усвоены методы организации взаимодействия объектов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на С++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. С++: 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 05.05.2022).

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 05.05.2022).

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

# Приложение А

Программная реализация алгоритмов для решения задачи представлена ниже.

## Файл Application.cpp

Листинг 1 – Application.cpp

|  |
| --- |
| #include "Application.h"  #include "Child2.h"  #include "Child3.h"  #include "Child4.h"  #include "Child5.h"  #include "Child6.h"  Application::Application (Root\* \_parent = nullptr) {  Root::rt->setParent(\_parent);  Root::rt->setName("rootObj");  Root::rt->kids.push\_back(rt);  }  int Application::launchApp() {  cout << "Object tree" << "\n";  this->printTree(0);  string com, p;  cin >> com;  while (com != "END") {  cin >> p;  Root\* obj = findByPath(p);  if (com == "FIND") {  cout << "\n" << p;  for (int i = 0; i < 5; i++) {  cout << " ";  }  if (obj) cout << "Object name: " << obj->getName();  else cout << "Object is not found";  }  if (com == "SET") {  if (obj) {  setCurObj(obj);  cout << "\nObject is set: " << obj->getName();  }  else cout << "Object is not found: " << obj-> getCurObj()->getName() << " " << p;  }  cin >> com;  }  return 0;  }  void Application::buildTreeFromRoot(){  string par, kid;  int n = 0;  cin >> par;  this->setName(par);  Root\* child = nullptr;  while (par != "endtree") {  if (n == 2) child = new Child2(kid, findByPath(par));  if (n == 3) child = new Child3(kid, findByPath(par));  if (n == 4) child = new Child4(kid, findByPath(par));  if (n == 5) child = new Child5(kid, findByPath(par));  if (n == 6) child = new Child6(kid, findByPath(par));  cin >> par;  if (par != "endtree") {  if (!findByPath(par)) {  cout << "Object tree" << "\n";  this->printTree(0);  cout << "\nThe head object " << par << " is not found";  exit(0);  }  cin >> kid >> n;  }  }  } |

## Файл Application.h

Листинг 2 – Application.h

|  |
| --- |
| #ifndef APPLICATION\_H  #define APPLICATION\_H  #include "Root.h"  class Application : public Root {  public:  Application(Root\* parent);  void buildTreeFromRoot();  int launchApp();  };  #endif |

## Файл Child2.h

Листинг 3 – Child2.h

|  |
| --- |
| #ifndef \_CHILD2\_H  #define \_CHILD2\_H  #include "Root.h"  class Child2 : public Root {  public:  Child2(string \_name, Root\* ptrToPar) : Root(\_name, ptrToPar) {};  using Root::Root;  };  #endif |

## Файл Child3.h

Листинг 4 – Child3.h

|  |
| --- |
| #ifndef \_CHILD3\_H  #define \_CHILD3\_H  #include "Root.h"  class Child3 : public Root {  public:  Child3(string \_name, Root\* ptrToPar) : Root(\_name, ptrToPar) {};  using Root::Root;  };  #endif |

## Файл Child4.h

Листинг 5 – Child4.h

|  |
| --- |
| #ifndef \_CHILD4\_H  #define \_CHILD4\_H  #include "Root.h"  class Child4 : public Root {  public:  Child4(string \_name, Root\* ptrToPar) : Root(\_name, ptrToPar) {};  using Root::Root;  };  #endif |

## Файл Child5.h

Листинг 6 – Child5.h

|  |
| --- |
| #ifndef \_CHILD5\_H  #define \_CHILD5\_H  #include "Root.h"  class Child5 : public Root {  public:  Child5(string \_name, Root\* ptrToPar) : Root(\_name, ptrToPar) {};  using Root::Root;  };  #endif |

## Файл Child6.h

Листинг 7 – Child6.h

|  |
| --- |
| #ifndef \_CHILD6\_H  #define \_CHILD6\_H  #include "Root.h"  class Child6 : public Root {  public:  Child6(string \_name, Root\* ptrToPar) : Root(\_name, ptrToPar) {};  using Root::Root;  };  #endif |

## Файл main.cpp

Листинг 8 – main.cpp

|  |
| --- |
| #include "Application.h"  using namespace std;  int main() {  Application app(nullptr);  app.buildTreeFromRoot();  return app.launchApp();  } |

## Файл Root.cpp

Листинг 9 – Root.cpp

|  |
| --- |
| #include "Root.h"  #include <iostream>  #include <sstream>  Root\* Root::rt = new Root();  Root::Root(){  parent = nullptr;  }  Root::Root(string \_name, Root\* ptrToPar) {  this -> name = \_name;  if (ptrToPar == nullptr) {  setParent(rt);  this -> parent -> kids.push\_back(this);  }  else {  this -> parent = ptrToPar;  ptrToPar -> kids.push\_back(this);  }  }  void Root::setName(string \_name) {  this->name = \_name;  }  string Root::getName() {  return name;  }  void Root::setParent(Root\* \_parent) {  if (this->parent) {  for (int i = 0; i < this->parent->kids.size(); i++) {  if (this->parent->kids[i] == this) {  this->parent->kids.erase(this->parent->kids.begin() + i);  break;  }  }  }  if (\_parent) {  this->parent = \_parent;  parent->kids.push\_back(this);  }  }  Root\* Root::getParent(){  return parent;  }  void Root::setStat(int num) {  if (num == 0) {  this->stat = 0;  if (this->kids.size() > 0) {  for (int i = 0; i < kids.size(); i++) {  kids[i]->setStat(0);  }  }  }  else {  if (this->getParent()) {  if (this->getParent()->getStat()) {  this->stat = 1;  }  else this->stat = 0;  }  else {  this->stat = 1;  }  }  }  bool Root::getStat() {  return stat;  }  void Root::setCurObj(Root\* \_obj) {  curObj = \_obj;  }  Root\* Root::getCurObj() {  return curObj;  }  Root\* Root::findByName(string n) {  Root\* objToFind = nullptr;  if (this->getName() == n) return this;  if (this->kids.size() > 0) {  for (int i = 0; i < kids.size(); i++) {  if (kids[i]->getName() == n) {  objToFind = kids[i];  }  if (!objToFind){  objToFind = kids[i]->findByName(n);  }  else return objToFind;  }  }  return objToFind;  }  vector<string> getNamesInPath(string path) {  vector <string> names;  string name;  stringstream str(path);  while (getline(str, name, '/')){  if (name.size() > 0){  names.push\_back(name);  }  }  return names;  }  void Root::printTree(int deep = 0) {  if (deep == 0) cout << this->getName();  else {  cout << "\n";  for (int i = 0; i < deep; i++) { cout << " "; }  cout << this->getName();  }  if (this->kids.size() > 0) {  for (int i = 0; i < kids.size(); i++) {  kids[i]->printTree(deep + 1);  }  }  }  Root\* Root::findByPath(string path) {  if (path[0] == '/') {  if (path.size() == 1) return this;  if (path[1] == '/') return findByName(path.substr(2));  return findByAbsPath(path);  }  if (path[0] == '.' && path[1]!= '/') return curObj;  return findByRelPath(path);  }  Root\* Root::findByAbsPath(string path) {  Root\* find = this;  vector<string> names = getNamesInPath(path);  for (int i = 0; i < names.size(); i++) {  Root\* prev = find;  for (int j = 0; j < find->kids.size(); j++) {  if (find->kids[j]->getName() == names[i]) {  find = find->kids[j];  break;  }  }  if (find == prev) return nullptr;  }  return find;  }  Root\* Root::findByRelPath(string path) {  Root\* find = curObj;  vector<string> names = getNamesInPath(path);  for (int i = 0; i < names.size(); i++) {  Root\* prev = find;  for (int j = 0; j < find->kids.size(); j++) {  if (find->kids[j]->getName() == names[i]) {  find = find->kids[j];  break;  }  }  if (find == prev) return nullptr;  }  return find;  }  void Root::printStatTree(int deep = 0) {  if (deep == 0) {  cout << "\n" << this->getName() << " ";  if (this->getStat()) cout << "is ready";  else cout << "is not ready";  }  else {  cout << "\n";  for (int i = 0; i < deep; i++) { cout << " "; }  cout << this->getName() << " ";  if (this->getStat()) cout << "is ready";  else cout << "is not ready";  }  if (this->kids.size() > 0) {  for (int i = 0; i < kids.size(); i++) {  kids[i]->printStatTree(deep + 1);  }  }  }  Root::~Root() {}; |

## Файл Root.h

Листинг 10 – Root.h

|  |
| --- |
| #ifndef \_ROOT\_H  #define \_ROOT\_H  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class Root {  private:  string name;  Root\* parent;  bool stat = false;  Root\* curObj = this;  public:  static Root\* rt;  vector <Root\*> kids;  Root();  Root(string \_name, Root\* ptrToPar);  void setName(string \_name);  string getName();  void setStat(int num);  bool getStat();  void setCurObj(Root\* \_obj);  Root\* getCurObj();  Root\* findByName(string n);  Root\* findByPath(string path);  Root\* findByAbsPath(string path);  Root\* findByRelPath(string path);  void printTree(int deep);  void printStatTree(int deep);  void setParent(Root\* parent);  Root\* getParent();  ~Root();  };  #endif |

# Приложение Б

Результат тестирования программы представлен в таблице 23.

Таблица Б – Результат тестирования программы

| Входные данные | Ожидаемые выходные данные | Фактические выходные данные |
| --- | --- | --- |
| root  / object\_1 3  / object\_2 2  /object\_2 object\_4 3  /object\_2 object\_5 4  / object\_3 3  /object\_2 object\_3 6  /object\_1 object\_7 5  /object\_2/object\_4 object\_7 3  endtree  FIND object\_2/object\_4  SET /object\_2  FIND //object\_5  FIND /object\_15  FIND .  FIND object\_4/object\_7  END | Object tree  root  object\_1  object\_7  object\_2  object\_4  object\_7  object\_5  object\_3  object\_3  object\_2/object\_4 Object name: object\_4  Object is set: object\_2  //object\_5 Object name: object\_5  /object\_15 Object is not found  . Object name: object\_2  object\_4/object\_7 Object name: object\_7 | Object tree  root  object\_1  object\_7  object\_2  object\_4  object\_7  object\_5  object\_3  object\_3  object\_2/object\_4 Object name: object\_4  Object is set: object\_2  //object\_5 Object name: object\_5  /object\_15 Object is not found  . Object name: object\_2  object\_4/object\_7 Object name: object\_7 |