

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#__RefHeading___Toc9484_3323595626)

[1 ПОСТАНОВКА ЗАДАЧИ 8](#__RefHeading___Toc9486_3323595626)

[1.1 Описание входных данных 9](#__RefHeading___Toc9488_3323595626)

[1.2 Описание выходных данных 11](#__RefHeading___Toc9490_3323595626)

[2 МЕТОД РЕШЕНИЯ 13](#__RefHeading___Toc9492_3323595626)

[3 ОПИСАНИЕ АЛГОРИТМОВ 18](#__RefHeading___Toc9494_3323595626)

[3.0 Алгоритм метода get\_object\_by\_coordinate класса base 18](#__RefHeading___Toc9496_3323595626)

[3.1 Алгоритм метода bild\_tree\_objects класса application 21](#__RefHeading___Toc9498_3323595626)

[3.2 Алгоритм метода exec\_app класса application 24](#__RefHeading___Toc9500_3323595626)

[3.3 Алгоритм функции main 26](#__RefHeading___Toc9502_3323595626)

[3.4 Алгоритм метода find\_object\_by\_name класса base 27](#__RefHeading___Toc9504_3323595626)

[3.5 Алгоритм метода print\_object\_tree класса base 29](#__RefHeading___Toc9506_3323595626)

[3.6 Алгоритм метода print\_state класса base 30](#__RefHeading___Toc9508_3323595626)

[3.7 Алгоритм метода install\_state класса base 32](#__RefHeading___Toc9510_3323595626)

[3.8 Алгоритм метода turn\_off\_state класса base 33](#__RefHeading___Toc9512_3323595626)

[3.9 Алгоритм метода get\_object\_by\_name класса base 33](#__RefHeading___Toc9514_3323595626)

[3.10 Алгоритм метода get\_root класса base 34](#__RefHeading___Toc9516_3323595626)

[3.11 Алгоритм конструктора класса base 35](#__RefHeading___Toc9518_3323595626)

[3.12 Алгоритм метода change\_parent класса base 36](#__RefHeading___Toc9520_3323595626)

[3.13 Алгоритм метода get\_parent класса base 38](#__RefHeading___Toc9522_3323595626)

[3.14 Алгоритм метода set\_state класса base 38](#__RefHeading___Toc9524_3323595626)

[3.15 Алгоритм метода get\_state класса base 39](#__RefHeading___Toc9526_3323595626)

[3.16 Алгоритм метода get\_child\_by\_name класса base 39](#__RefHeading___Toc9528_3323595626)

[3.17 Алгоритм деструктора класса base 40](#__RefHeading___Toc9530_3323595626)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 42](#__RefHeading___Toc9532_3323595626)

[5 КОД ПРОГРАММЫ 61](#__RefHeading___Toc9534_3323595626)

[5.0 Файл application.cpp 61](#__RefHeading___Toc9536_3323595626)

[5.1 Файл application.h 62](#__RefHeading___Toc9538_3323595626)

[5.2 Файл base.cpp 63](#__RefHeading___Toc9540_3323595626)

[5.3 Файл base.h 66](#__RefHeading___Toc9542_3323595626)

[5.4 Файл cl\_1.cpp 67](#__RefHeading___Toc9544_3323595626)

[5.5 Файл cl\_1.h 67](#__RefHeading___Toc9546_3323595626)

[5.6 Файл cl\_2.cpp 67](#__RefHeading___Toc9548_3323595626)

[5.7 Файл cl\_2.h 67](#__RefHeading___Toc9550_3323595626)

[5.8 Файл cl\_3.cpp 68](#__RefHeading___Toc9552_3323595626)

[5.9 Файл cl\_3.h 68](#__RefHeading___Toc9554_3323595626)

[5.10 Файл cl\_4.cpp 68](#__RefHeading___Toc9556_3323595626)

[5.11 Файл cl\_4.h 69](#__RefHeading___Toc9558_3323595626)

[5.12 Файл cl\_5.cpp 69](#__RefHeading___Toc9560_3323595626)

[5.13 Файл cl\_5.h 69](#__RefHeading___Toc9562_3323595626)

[5.14 Файл cl\_6.cpp 69](#__RefHeading___Toc9564_3323595626)

[5.15 Файл cl\_6.h 70](#__RefHeading___Toc9566_3323595626)

[5.16 Файл main.cpp 70](#__RefHeading___Toc9568_3323595626)

[6 ТЕСТИРОВАНИЕ 71](#__RefHeading___Toc9570_3323595626)

[ЗАКЛЮЧЕНИЕ 72](#__RefHeading___Toc9572_3323595626)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 73](#__RefHeading___Toc9574_3323595626)

# ВВЕДЕНИЕ

Объектно-ориентированное программирование (ООП) в наши дни это одна из самых популярных парадигм программирования,  основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

ООП это максимально простой и понятный для человека способ описать окружающий мир и построить работу программы по подобию реального мира, она позволяет структурировать информацию с точки зрения управляемости, что существенно улучшает  управляемость самим процессом моделирования , что,  в свою очередь, особенно важно при создании крупных проектов.

Основные принципы структурирования в случае ООП связаны с различными  аспектами базового  понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью :

Инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды "что делать",  без уточнения, как именно делать;

Наследование для быстрой и безопасной организации родстенных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя все остальное, учтённое в предидущих шагах;

Полфирмизм для определения точки, в которой удиное управление лучше разделить или наоборот - собрать воедино;

Язык C++ также прекрасно подходит для изучения ООП благодаря тому, что он сочетает в себе высокоуровневые и низкоуровневые средства. Использование таких средств, как указатели и динамическое выделение памяти, позволяет понять (или в дальнейшем способствует пониманию), что такое стек, куча, стек вызовов, раскрутка и т.д. Помимо этого, на практике понимание консепции адрессов и адресной арифметики. На примерах демонстрируется, что память надо выделять и освобождать, что существуют утечки памяти. Чётко разграниченные уровни доступа к членам класса, возможность множественного наследования и динамический полиморфизм дают возможность быстро усвоить основне концепции ООП. Так-же C++ не ограничивает использованием только ООП, позволяет понимать функциональное программирование, что учит сочетать разные средства разработки, ведь ООП не может существовать полностью без процедурного программирования

# 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 задачи используются:

• Объект стандартного потока вывода на экран cout;

• Объект стандартного потока ввода с клавиатуры cin;

• Стандартная функция substr(pos, count) библиотеки <string> -возвращает подстроку данной строки начиная с символом с индексомpos количеством count или до конца строки.

• Объект ob\_application класса application;

• Объекты классов cl\_2, cl\_3, cl\_4, cl\_5, cl\_6 (имена и количествозадаются пользователем);

• Класс base:

Свойства/поля:

• Поле, отвечающее за наименование объекта

• Наименование - object\_name;

• Тип - строковый;

• Модификатор доступа - private.

• Поле, отвечающее за указатель на головной объект для текущего объекта .

• Наименование - p\_parent;

• Тип - указатель на объект класса base (или его наследников);

• Модификатор доступа - private.

• Поле, отвечающее за состояние объекта

• Наименование - state;

• Тип - целочисленный;

• Модификатор доступа - private.

• Поле, отвечающее за массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии:

• Наименование - children;

• Тип - вектор ;

• Модификатор доступа - priva

• методы:

• Конструктор base:

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта (имеет значение по умолчанию).

• set\_object\_name:

• Функционал - используется для установки имени объекта.

• get\_object\_name:

• Функционал - используется для получения имени объекта.

• change\_parent:

• Функционал - используется для переопределения головного объекта для текущего в дереве иерархии.

• get\_parent:

• Функционал - используется для получения указателя на головной объект текущего объекта.

• set\_state:

• Функционал - используется для определения состояние объекта.

• get\_state:

• Функционал - используется для получения состояние объекта.

• get\_child\_by\_name:

• Функционал - используется для получения указателя объект-потомок по имени объекта

• print\_object\_tree:

• Функционал - используется для вывода наименований

объектов в дереве иерархии слева направо и сверху вниз.

• Деструктор ~base:

• Функционал - деструктор, удаление объектов иерархического дерева.

• get\_object\_by\_coordinate()

• Функционал - получения указателя на любой объект всоставе дерева иерархии объектов согласно пути(координаты).

• Класс applicatoin

• методы:

• Конструктор application:

• Функционал - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии

• bild\_tree\_objects ()

• Функционал - построение дерева иерархии объектов.

• exec\_app()

• Функционал - метод запуска приложения (начало функционирования системы, выполнения алгорит марешения задачи).

• Класс cl\_1 (наследуетcя от base):

• Методы

• Конструктор cl\_1

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

• Класс cl\_2 (наследуется от base):

• Методы:

• Конструктор cl\_2()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

• Класс cl\_3 (наследуется от base):

• Методы:

• Конструктор cl\_3()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

• Класс cl\_4 (наследуется от base):

• Методы:

• Конструктор cl\_4()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

• Класс cl\_5 (наследуется от base):

• Методы:

• Конструктор cl\_5()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

• Класс cl\_6(наследуется от base):

• Методы:

• Конструктор cl\_6()

• Функционал - параметризированный конструктор с параметрами указателя на головной объект в дереве иерархии и наименованием объекта.

# 3 ОПИСАНИЕ АЛГОРИТМОВ

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

## Алгоритм метода get\_object\_by\_coordinate класса base

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

Параметры: строковый, path - путь(координата) объекта.

Возвращаемое значение: указатель на объект класса класса base.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода get\_object\_by\_coordinate класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Значение  параметра path  равно "/" | Возврат результата  вызова метода get\_root() | ∅ |
| Значение  параметра path  равно "." | Возврат значения  указателя на текущий  объект | ∅ |
| Два первых  символа параметра  path равны "//" | Возврат результата  вызова метода  get\_object\_by\_name() от  корневого объекта с  параметром значения  подстроки path, начиная  с третьего символа | ∅ |
| Первый символ  параметра path  равен '/' | Возврат результата  вызова метода  get\_object\_by\_coordinate  () для корневого  объекта с параметром  значения подстроки,  начиная со второго  символа | ∅ |
|  |  | 2 |
| 2 |  | Инициализация  целочисленной  переменной index  значением равным  размерности строки  path | 3 |
| 3 |  | Инициализация  целочисленной  переменной счетчика i  значением 0 | 4 |
| 4 | Значение  переменной i  меньше  размерности  строки path |  | 5 |
|  |  | 6 |
| 5 | Значение i - го  элемента строки  равно '/' | Присвоение  переменной index  значения переменной i | 6 |
|  | Увеличение  переменной счетчика i  на 1 | 4 |
| 6 |  | Инициализация  строковой переменной s  значением подстроки с  первого элемента  длиной index | 7 |
| 7 |  | Инициализация  целочисленной переменной счетчика i  значением 0 | 8 |
| 8 | значение  переменной i  меньше размера  массива children |  | 9 |
|  | Возврат нулевого  указателя | ∅ |
| 9 | Результат вызова  метода  get\_object\_name  подчиненного  объекта равен  значению  переменной s |  | 10 |
|  |  | 11 |
| 10 | Размер строки path  равен значению  переменной index | Возврат указателя на  подчиненный объект  для текущего | ∅ |
|  | Возврат результата  вызова метода  get\_object\_by\_name()  подчиненного объекта к  текущему с параметром  значения подстроки  path, начиная с символа  index+1 | ∅ |
| 11 |  | Увеличение  переменной счетчика i  на 1 | 8 |

## Алгоритм метода bild\_tree\_objects класса application

Функционал: Постраение дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода bild\_tree\_objects класса application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление строковых  переменных string,path | 2 |
| 2 |  | Объявление  целочисленной  переменной number | 3 |
| 3 |  | Ввод с клавиатуры  значения переменной  name | 4 |
| 4 |  | Вызов метода  set\_object\_name()  текущего объекта с  значением параметра  name | 5 |
| 5 |  | Объявление указателя  obj на объект класса base | 6 |
| 6 | Осуществляется  ввод данных |  | 7 |
|  |  | ∅ |
| 7 | Значение  переменной path =  "endtree" |  | ∅ |
|  | Ввод с клавиатуры  значения переменных  name,number | 8 |
| 8 | Результат вызова  метода  get\_object\_by\_coordinate с параметром path равен 0 | вызов метода  print\_object\_tree с  параметрами указателя  на текущий объект и 0 | 9 |
|  |  | 11 |
| 9 |  | Вывод перехода на  новую строку + "The  head object + path +" is  not found" | 10 |
| 10 |  | завершение программы | ∅ |
| 11 | Значения  переменной  number = 2 | Coздание объекта класса  cl\_2 посредством  параметризированного  конструктора | 6 |
|  |  | 12 |
| 12 | Значения  переменной  number = 3 | Coздание объекта класса  cl\_3 посредством  параметризированного  конструктора | 6 |
|  |  | 13 |
| 13 | Значения  переменной  number = 4 | Coздание объекта класса  cl\_4 посредством  параметризированного  конструктора | 6 |
|  |  | 14 |
| 14 | Значения  переменной  number = 5 | Coздание объекта класса  cl\_5 посредством  параметризированного  конструктора | 6 |
|  |  | 15 |
| 15 | Значения  переменной  number = 6 | Coздание объекта класса  cl\_6 посредством  параметризированного  конструктора | 6 |
|  |  | 6 |

## Алгоритм метода exec\_app класса application

Функционал: метод запуска приложения (начало функционирования системы,выполнения алгоритма решения задачи).

Параметры: нет.

Возвращаемое значение: целочисленное, 0 - при успешном выполнении, вдругом случае - код ошибки.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода exec\_app класса application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод "Object tree" | 2 |
| 2 |  | вызов метода  print\_object\_tree с  параметрами указателя на  текущий объект и 0 | 3 |
| 3 |  | Объявление строковых  переменных  command,path | 4 |
| 4 |  | Инициализация указателя  р на класс base значением  указателя на текущий  объект | 5 |
| 5 | конец входных данных |  | ∅ |
|  |  | 6 |
| 6 | Значение  переменной  command =  "END" |  | ∅ |
|  | ввод значения  переменной path | 7 |
| 7 | Значение  переменной  command =  "SET" |  | 8 |
|  |  | 10 |
| 8 | Результат вызова метода  get\_object\_by\_co  ordinate() по  указателю р не  равен 0 | Присвоение указателю p результата вызова метода  get\_object\_by\_coordinate()  по указателю р | 9 |
|  | Вывод перехода на новую  строку +"Object is not  found: "+ результат  вызова метода  get\_object\_name() по  указателю р+ значения  переменной path | 5 |
| 9 |  | Вывод перехода на новую  строку + "Object is set: " +  результат вызова метода  get\_object\_name() по  указателю р | 5 |
| 10 | Значение  переменной  command =  "FIND" |  | 11 |
|  |  | 5 |
| 11 | Результат вызова  метода  get\_object\_by\_co  ordinate() по  указателю р не  равен 0 | вывод перехода на новую  строку+значения  переменной path + "Object  is found: "+результат  вызова метода  get\_object\_name с  координатой path | 5 |
|  |  | 5 |

## Алгоритм функции main

Функционал: основаная функция.

Параметры: нет.

Возвращаемое значение: целочисленное, результат выполнения программы.

Алгоритм функции представлен в таблице 4.

Таблица 4 – Алгоритм функции main

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Создание объекта ob\_application  класса application посредством  параметризированного  конструктора с параметром  нулевого указателя | 2 |
| 2 |  | вызов метода bild\_tree\_objects  для объекта ob\_application | 3 |
| 3 |  | возврат результата вызова  метода exec\_app для объект  ob\_applicaton | ∅ |

## Алгоритм метода find\_object\_by\_name класса base

Функционал: Поиск объекта на дереве иерархии по имени (метод возвращаетуказатель на найденный объект или nullptr).

Параметры: строковый, name - наименование объекта.

Возвращаемое значение: указатель на объект класса base.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода find\_object\_by\_name класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация  указателя p на  объект класса base  значением нулевого  указателя | 2 |
| 2 | Результат вызова  метода  get\_object\_name для  текущего объекта  равен значению  параметра name | вовращение  указателя на  текущий объект | ∅ |
|  | Указателю p  присвоить значение  указателя на  текущий объект | 3 |
| 3 |  | Инициализация  целочисленной  переменной  счетчика i значением  0 | 4 |
| 4 | Значение переменной i  меньше размера  массива children |  | 5 |
|  |  | 7 |
| 5 | Результат вызова  метода  get\_object\_name для  объекта подчиненного  текущему с именем  name равно значению  параметра name | возврат результата  вызова метода  find\_object\_by\_name  для объекта  подчиненного  текущему | ∅ |
|  |  | 6 |
| 6 |  | Увеличение  переменной  счетчика i на 1 | 4 |
| 7 | Указатель на головной  объект для текущего  равен нулевому и  результат вызова  метода  get\_object\_name не  равен значению  параметра name | возврат нулевого  указателя | ∅ |
|  | возврат значение  указателя p | ∅ |

## Алгоритм метода print\_object\_tree класса base

Функционал: вывод дерева иерархии объектов.

Параметры: указатель на класс base, parent - указатель на головной объект;целочисленный, level - уровень вывода объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода print\_object\_tree класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление строковой  переменной s | 2 |
| 2 | Значение  параметра level  больше нуля | Добавление к строке s  количество пробелов  равное level\*4 | 3 |
|  |  | 3 |
| 3 |  | вывод переменной s+  результата вызова метода  get\_object\_name+переход  на новую строку | 4 |
| 4 | размер массива  children равен 0 |  | ∅ |
|  |  | 5 |
| 5 |  | Инициализация  целочисленной  переменной счетчика i  значением 0 | 6 |
| 6 | Значение  переменной i  меньше размера  массива children | вызов метода  print\_object\_tree с  параметрами значения  указателя на  подчиненный объект и  level+1 | 7 |
|  |  | ∅ |
| 7 |  | Увеличение переменной счетчика i на 1 | 6 |

## Алгоритм метода print\_state класса base

Функционал: вывод дерева иерархии объектов и отметок их готовности.

Параметры: указатель на класс base, parent - указатель на головной объект;целочисленный, level - уровень вывода объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода print\_state класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление строковой  переменной s | 2 |
| 2 | Значение  параметра level  больше нуля | Добавление к строке s  количество пробелов  равное level\*4 | 3 |
|  |  | 3 |
| 3 |  | вывод перехода на новую  строку+переменной s+  результата вызова метода  get\_object\_name+пробел | 4 |
| 4 | Результат вызова  метода get\_state  не равен 0 | вывод "is ready" | 5 |
|  | вывод "is not ready" | 5 |
| 5 | размер массива  children равен 0 |  | ∅ |
|  |  | 6 |
| 6 |  | Инициализация  целочисленной  переменной счетчика i  значением 0 | 7 |
| 7 | Значение  переменной i  меньше размера  массива children | вызов метода print\_state с  параметрами значения  указателя на  подчиненный объект и  level+1 | 8 |
|  |  | ∅ |
| 8 |  | Увеличение переменной  счетчика i на 1 | 7 |

## Алгоритм метода install\_state класса base

Функционал: установка готовности объекта.

Параметры: целочисленный, \_state - состояние готовности объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода install\_state класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Значение параметра  \_state не равно 0 |  | 2 |
|  |  | 4 |
| 2 | Значение указателя  на головной объект  для текущего равен  нулевому | Присвоение полю  state текущего  объекта значение  параметра \_state | ∅ |
|  |  | 3 |
| 3 | Значение поля state  для головного  объекта не равно 0 | Присвоение полю  state текущего  объекта значение  параметра \_state | ∅ |
|  |  | ∅ |
| 4 | Значение поля state  для текущего  объекта не равно 0 | вызов метода  turn\_off\_state | ∅ |
|  |  | ∅ |

## Алгоритм метода turn\_off\_state класса base

Функционал: Функцинал - отключение готовности для головного объекта иего подчиненных.

Параметры: HeT.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода turn\_off\_state класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение полю state  для текущего объекта  значение 0 | 2 |
| 2 | размер массива  children равен 0 |  | ∅ |
|  |  | 3 |
| 3 |  | Инициализация  целочисленной  переменной счетчика i  значением 0 | 4 |
| 4 | Значение  переменной i  меньше размера  массива children | вызов метода  turn\_off\_state для  объекта подчиненного к  текущему | 5 |
|  |  | ∅ |
| 5 |  | Увеличение переменной  счетчика i на 1 | 4 |

## Алгоритм метода get\_object\_by\_name класса base

Функционал: получение значения указателя на объект по имени.

Параметры: строковый, name - наименование объекта.

Возвращаемое значение: указатель на объект класса base.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get\_object\_by\_name класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | возврат результата метода  find\_object\_by\_name c  параметром name для корневого  объекта | ∅ |

## Алгоритм метода get\_root класса base

Функционал: Получение указателя на головной объект.

Параметры: нет.

Возвращаемое значение: указатель на объект класса base(указатель накорневой объект).

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода get\_root класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация указателя  p на класс base значением  указателя на текущий  объект | 2 |
| 2 | Значение  указателя на  головной объект  не равно 0 | Присвоение указателю р  значение указателя на  головной объект | 2 |
|  |  | 3 |
| 3 |  | возврат значения p | ∅ |

## Алгоритм конструктора класса base

Функционал: параметризированный конструктор, устанавливается имяобъекта и головной объект для текущего.

Параметры: p\_parent - указатель на объект класса base (указатель на головнойобъект в дереве иерархии); object\_name - строковый, наименование объекта.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение полю  object\_name текущего  объекта значение  параметра object\_name | 2 |
| 2 |  | Присвоение полю  p\_parent текущего  объекта значение  параметра p\_parent | 3 |
| 3 |  | Присвоение полю state  текущего объекта  значения 0 | 4 |
| 4 | Указатель на  головной объект  p\_parent для  текущего объекта  не нулевой | Добавление для  головного объекта  текущий объект в  массив подчиненных  объектов | ∅ |
|  |  | ∅ |

## Алгоритм метода change\_parent класса base

Функционал: используется для переопределения головного объекта длятекущего в дереве иерархии.

Параметры: .

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода change\_parent класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Указатель на  головной объект для  текущего объекта  равен нулевому  указателю или  значение параметра  new\_parent равно  нулевому указателю |  | ∅ |
|  |  | 2 |
| 2 |  | Инициализация  целочисленной  переменной счетчика  i значением 0 | 3 |
| 3 | Значение  переменной счетчика  меньше размера  массива  подчиненных  объектов children для  текущего головного |  | 4 |
|  |  | 5 |
| 4 | Значение i-ого  элемента массива  children текущего  головного равно  указателю на  текущий объект | удаление текущего  объекта из списка  наследников объектародителя | 5 |
|  | увеличение  переменной счетчика  i на 1 | 3 |
| 5 |  | Присвоение полю  p\_parent текущего  объекта значения  параметра new\_parent | 6 |
| 6 |  | добавление текущего  объекта в массив  наследников нового  головного объекта  new\_parent | ∅ |

## Алгоритм метода get\_parent класса base

Функционал: используется для получения указателя на головной объект длятекущего объекта.

Параметры: нет.

Возвращаемое значение: указатель на объект класса base (указатель наголовной объект для текущего объекта).

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода get\_parent класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | возврат указателя p\_parent на  головной объект для текущего  объекта | ∅ |

## Алгоритм метода set\_state класса base

Функционал: определение состояния объекта.

Параметры: целочисленный, state, состояние объекта.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода set\_state класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присвоение полю state текущего  объекта значение параметра state | ∅ |

## Алгоритм метода get\_state класса base

Функционал: используется для получения состояние объекта.

Параметры: нет.

Возвращаемое значение: целочисленное, состояние объекта.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода get\_state класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | возврат поля state текущего  объекта | ∅ |

## Алгоритм метода get\_child\_by\_name класса base

Функционал: Получение указателя на подчиненный объект по имени объекта.

Параметры: строковый, object\_name, наименование объекта.

Возвращаемое значение: указатель на объект класса base (указатель наподчиненный объект).

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода get\_child\_by\_name класса base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Размер массива  подчиненных  объектов children  равен 0 |  | ∅ |
|  | Инициализация  целочисленной  переменной счетчика  i значением 0 | 2 |
| 2 | Значение переменной счетчика i меньше  размера массива  children |  | 3 |
|  |  | ∅ |
| 3 | Имя объекта i-ого  элемента массива  children равен  значению параметра  object\_name | возврат i-ого  элемента массива  children | ∅ |
|  | Увеличение  переменной счетчика  i на 1 | 2 |

## Алгоритм деструктора класса base

Функционал: дестркутор.

Параметры: нет.

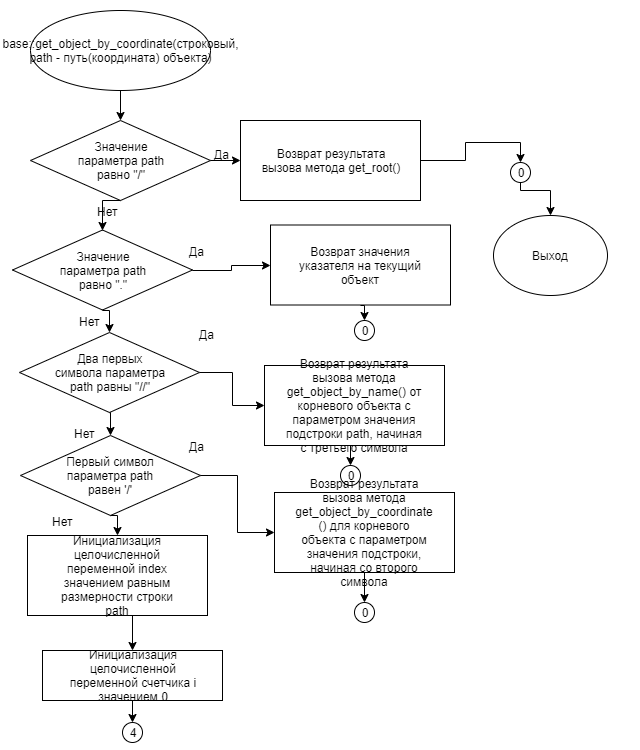
Алгоритм деструктора представлен в таблице 18.

Таблица 18 – Алгоритм деструктора класса base

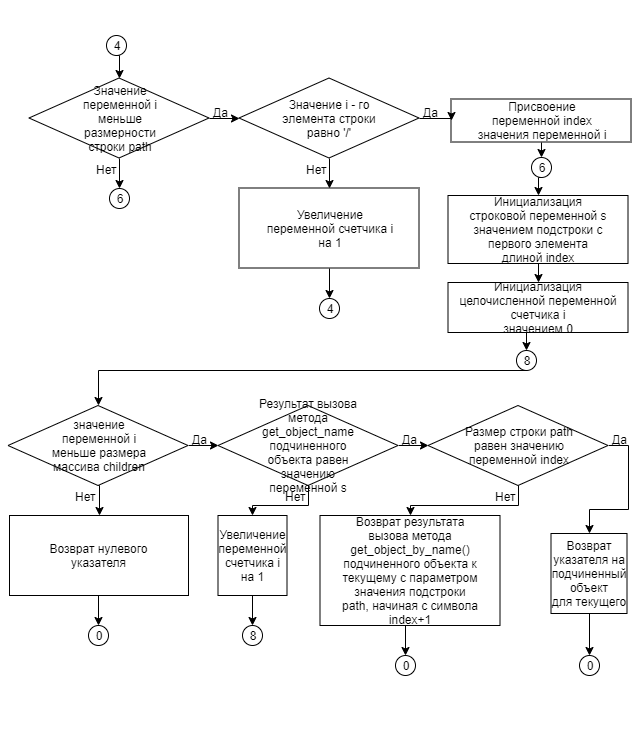
| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация  целочисленной  переменной счетчика i  значением 0 | 2 |
| 2 | Значение  переменной  счетчика i меньше  размера массива  children | Удаление значения iого элемента массива  children | 3 |
|  |  | ∅ |
| 3 |  | увеличение переменной счетчика i на 1 | 2 |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

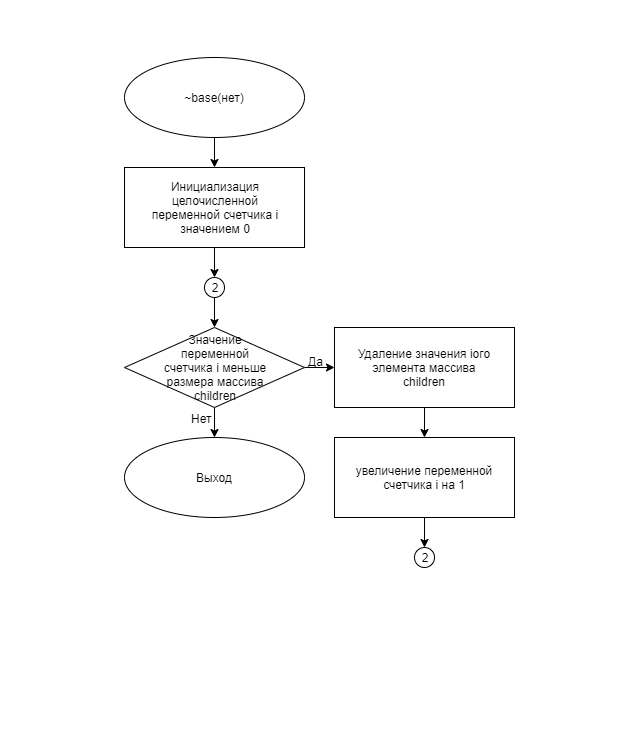
Представим описание алгоритмов в графическом виде на рисунках 1-19.



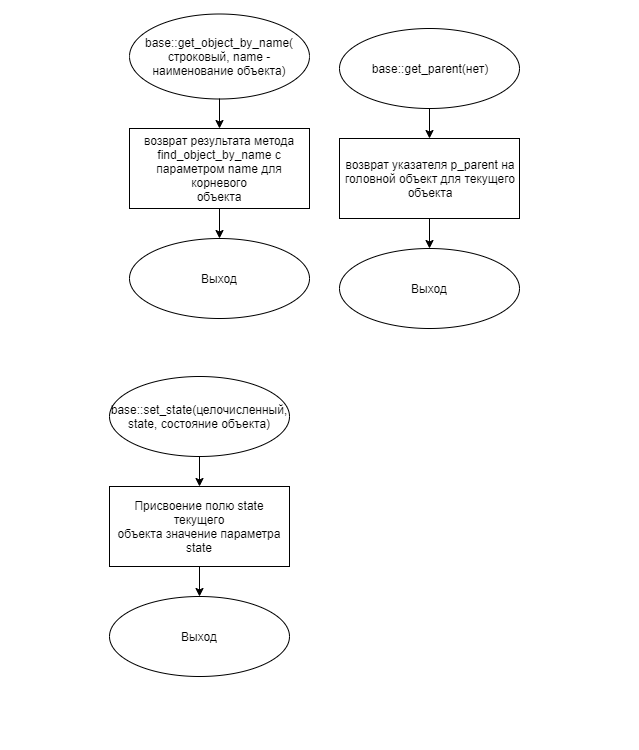
**Рисунок 1 – Блок-схема алгоритма**



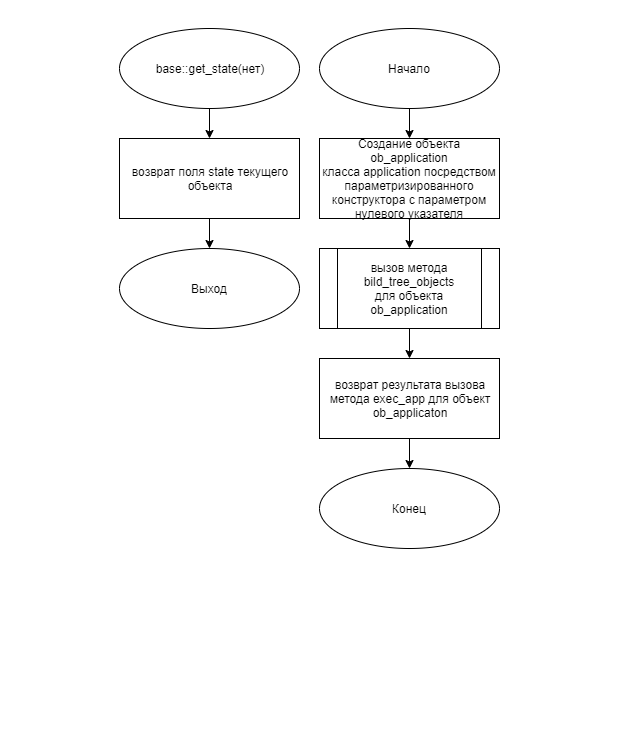
**Рисунок 2 – Блок-схема алгоритма**



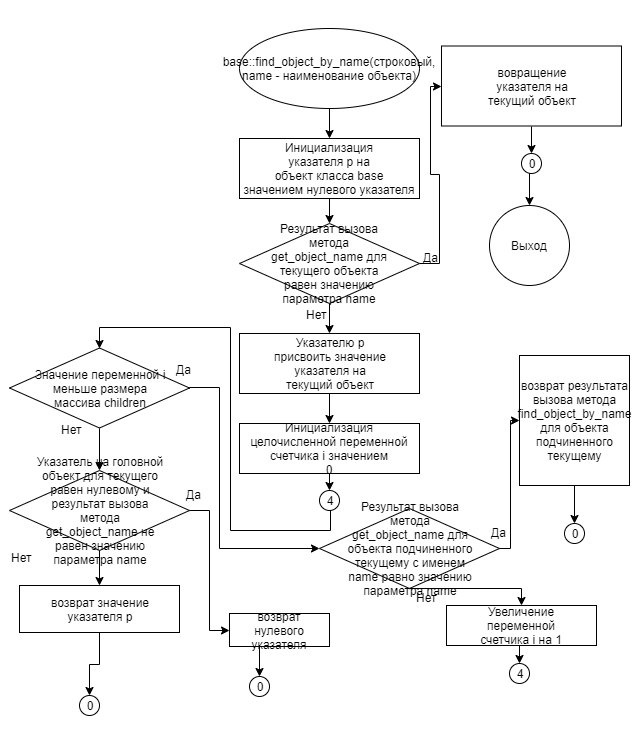
**Рисунок 3 – Блок-схема алгоритма**



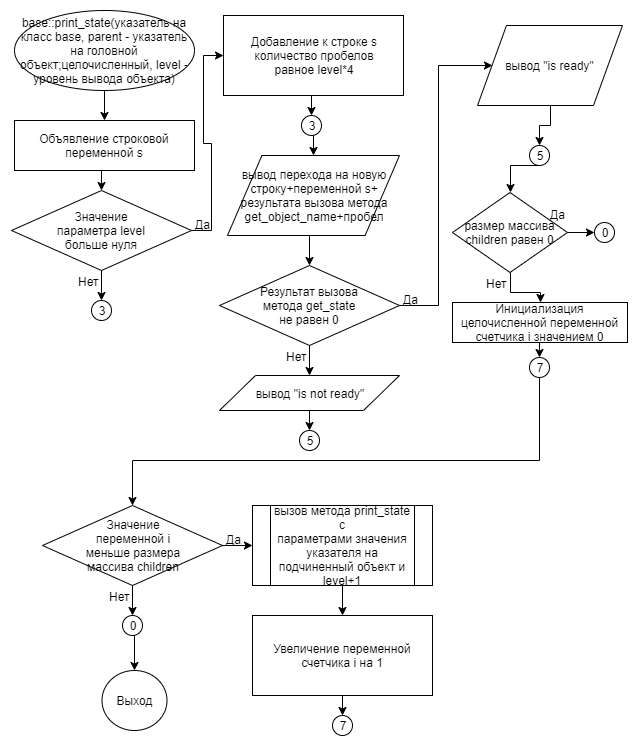
**Рисунок 4 – Блок-схема алгоритма**



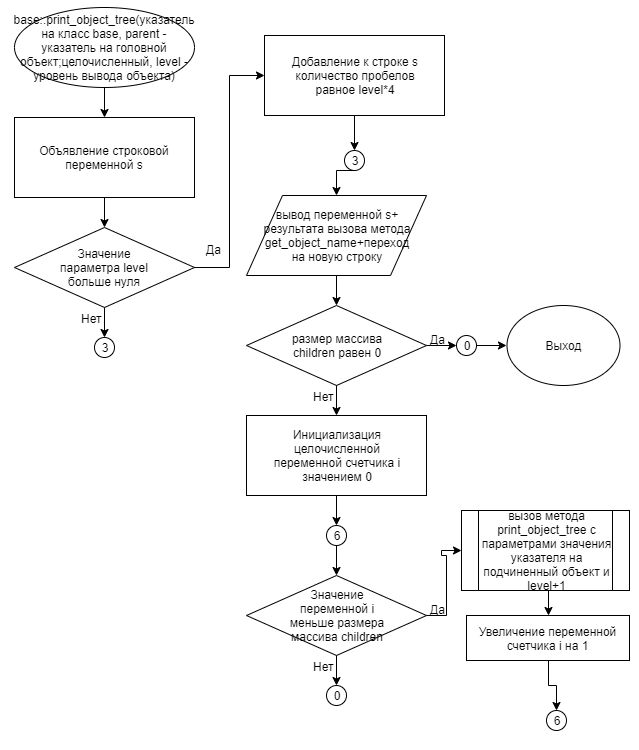
**Рисунок 5 – Блок-схема алгоритма**



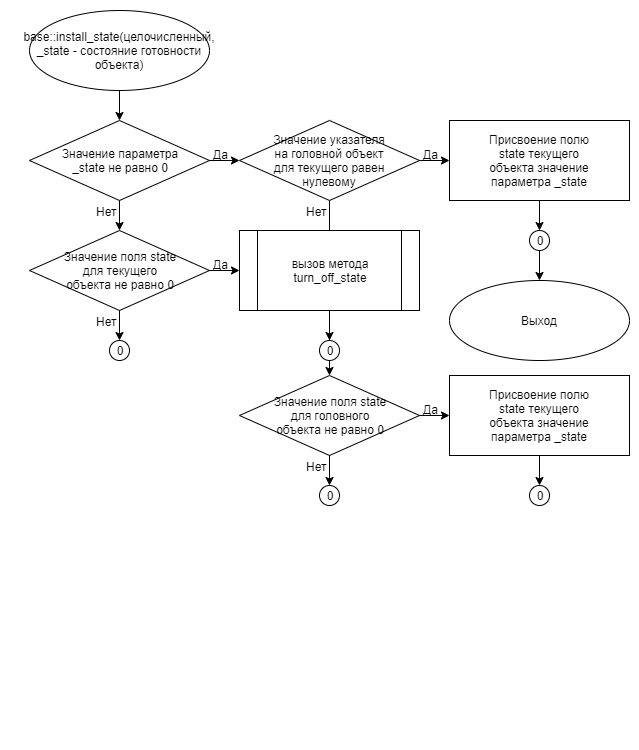
**Рисунок 6 – Блок-схема алгоритма**



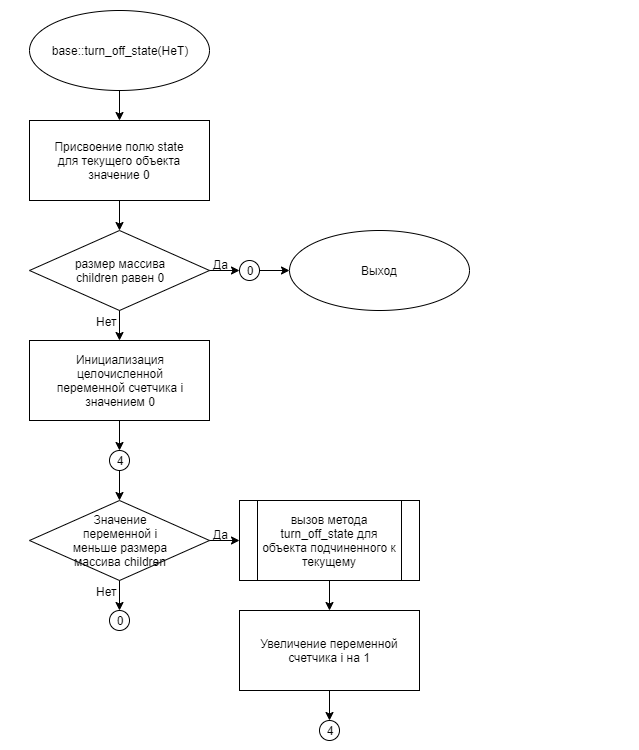
**Рисунок 7 – Блок-схема алгоритма**



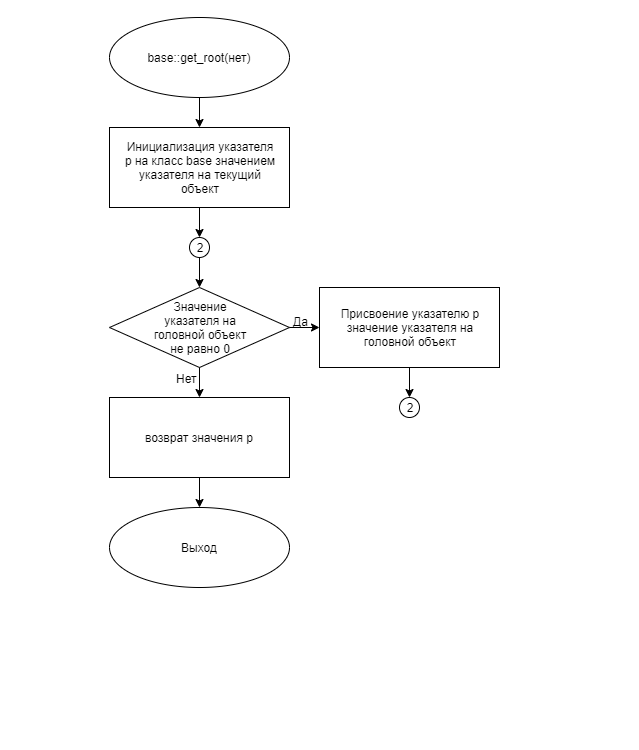
**Рисунок 8 – Блок-схема алгоритма**



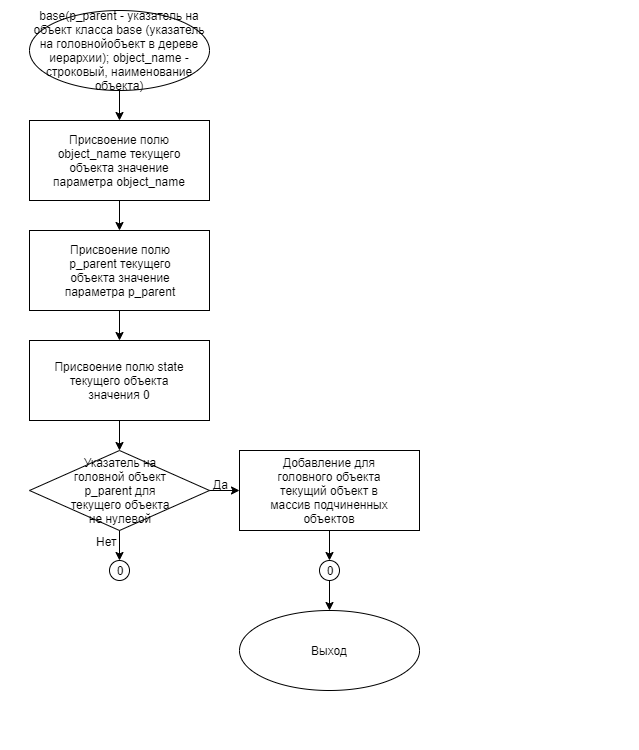
**Рисунок 9 – Блок-схема алгоритма**



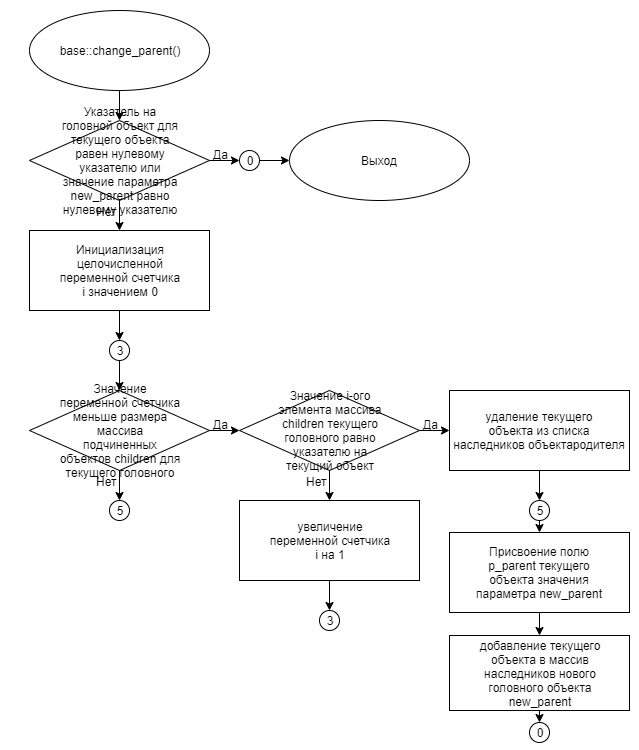
**Рисунок 10 – Блок-схема алгоритма**



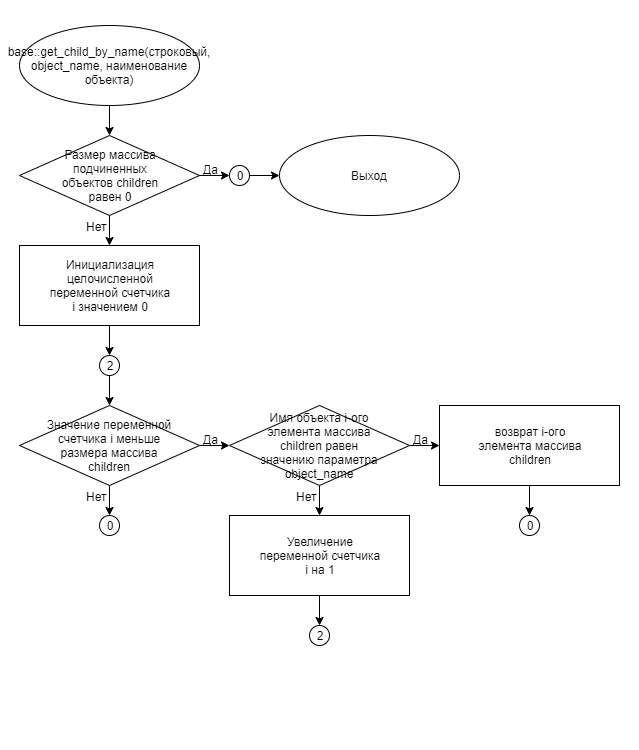
**Рисунок 11 – Блок-схема алгоритма**



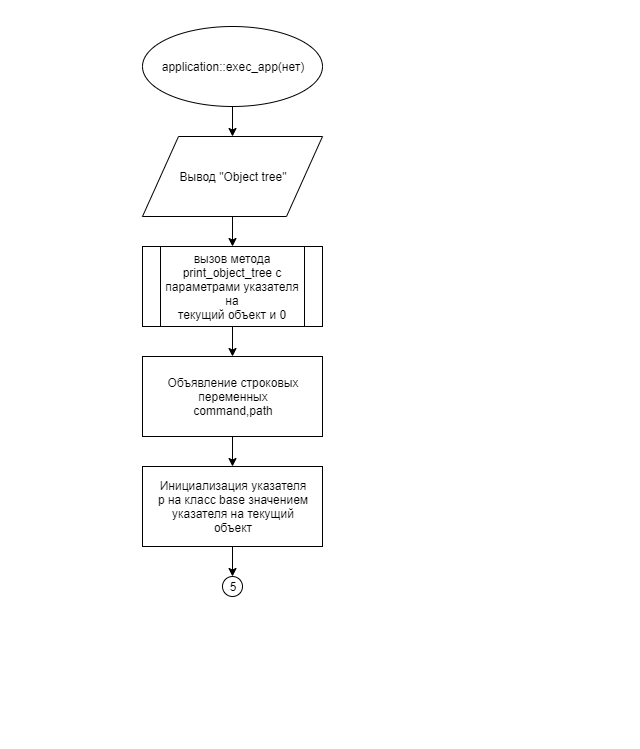
**Рисунок 12 – Блок-схема алгоритма**



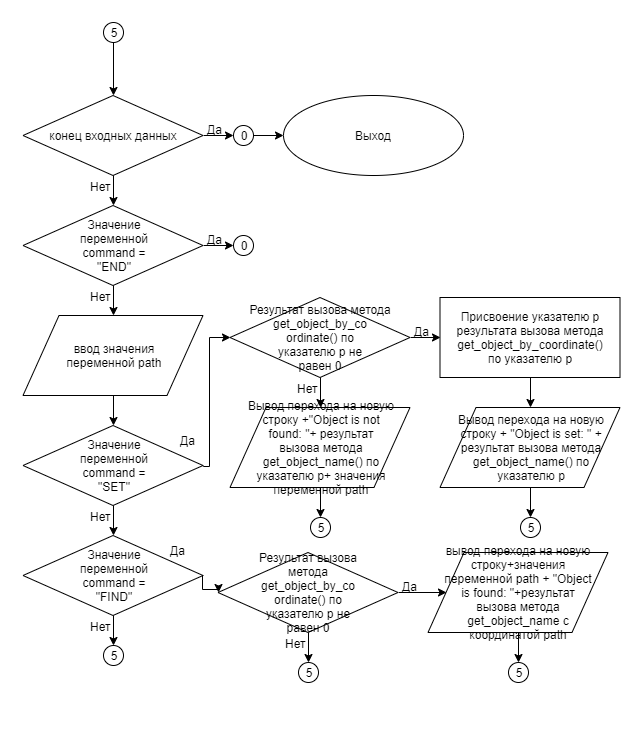
**Рисунок 13 – Блок-схема алгоритма**



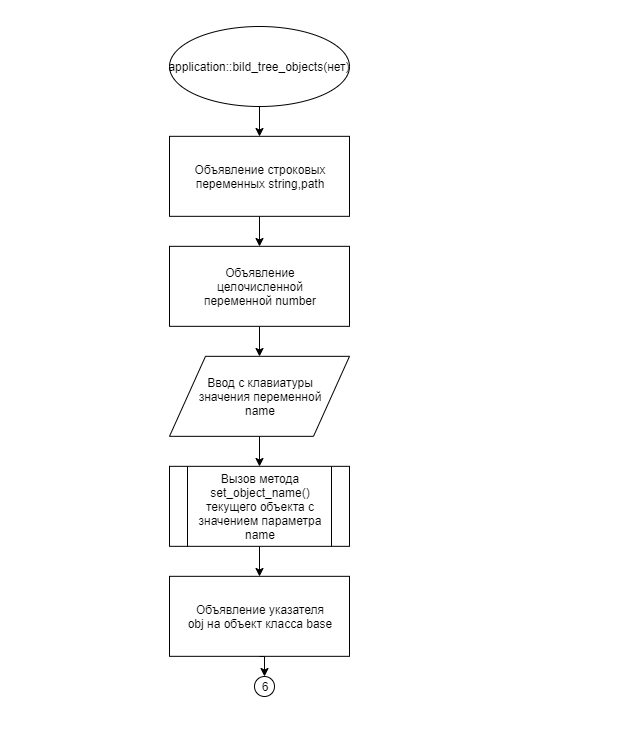
**Рисунок 14 – Блок-схема алгоритма**



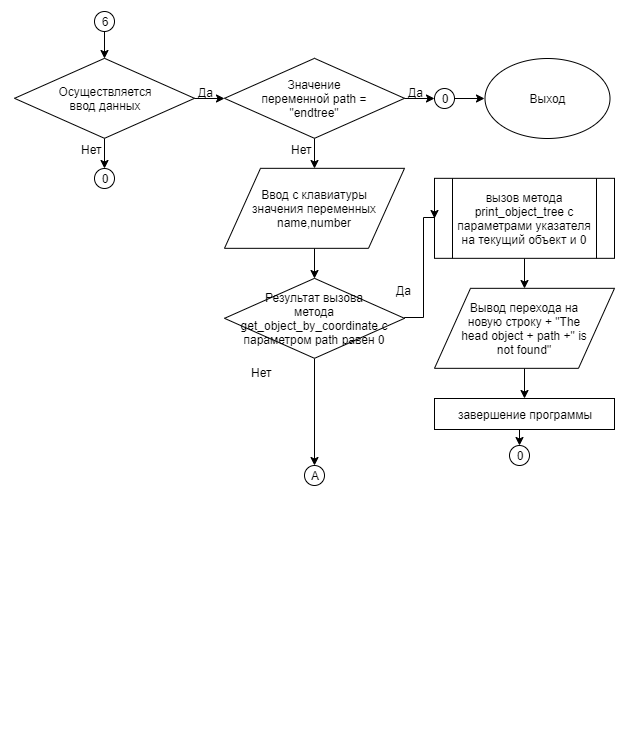
**Рисунок 15 – Блок-схема алгоритма**



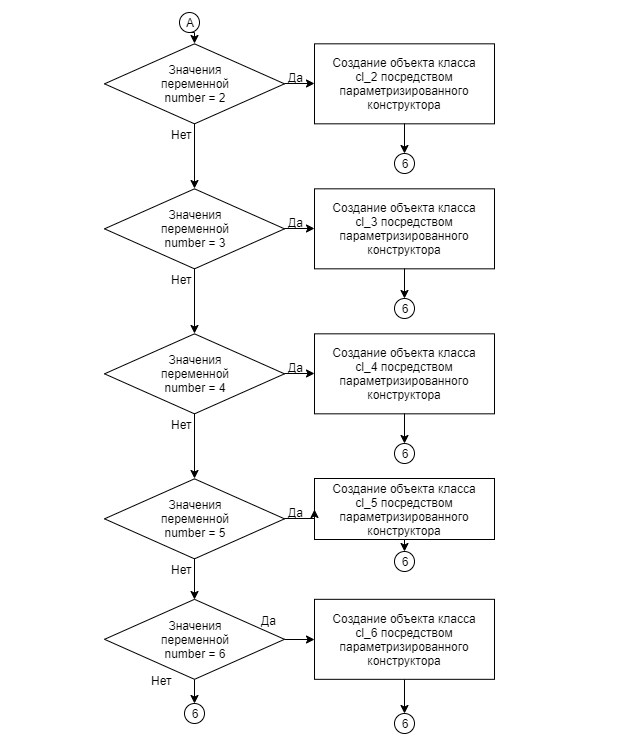
**Рисунок 16 – Блок-схема алгоритма**



**Рисунок 17 – Блок-схема алгоритма**



**Рисунок 18 – Блок-схема алгоритма**



**Рисунок 19 – Блок-схема алгоритма**

# 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

## **Файл** **application.cpp**

Листинг 1 – application.cpp

|  |
| --- |
| #include "application.h"  application::application(base\* p\_parent):base(p\_parent)  {}  void application:: bild\_tree\_objects()  {  string path, name;  int number;  cin>>name;  this->set\_object\_name(name);  base\* obj;  string s;  while (cin>>path)  {  if (path == "endtree")  return ;  if (!get\_object\_by\_coordinate(path))  {  cout<<"Object tree"<<endl;  print\_object\_tree(this,0);  cout <<endl<<"The head object "<< path <<" is not found";  exit(0);  }  else  {  cin>>name>>number;  if (number == 2) obj = new cl\_2(get\_object\_by\_coordinate(path), name);  else if (number == 3) obj = new cl\_3(get\_object\_by\_coordinate(path), name);  else if (number == 4) obj = new cl\_4(get\_object\_by\_coordinate(path), name);  else if (number == 5) obj = new cl\_5(get\_object\_by\_coordinate(path), name);  else if (number == 6) obj = new cl\_6(get\_object\_by\_coordinate(path), name);  }  }  }  int application::exec\_app()  {  cout<<"Object tree"<<endl;  print\_object\_tree(this,0);  string command,path;  base\* p = this;  while (cin>>command)  {  if (command == "END")  return 0;  cin>>path;  if (command == "SET")  {  if (p-> get\_object\_by\_coordinate(path))  {  p = p->get\_object\_by\_coordinate(path);  cout<<endl<<"Object is set: "<<p->get\_object\_name();  }  else  cout<<endl<<"Object is not found: "<<p->get\_object\_name()<<' '<<path;  }  if (command == "FIND")  {  if (p-> get\_object\_by\_coordinate(path))  cout<<endl<<path<<" Object name: "<<p-> get\_object\_by\_coordinate(path)->get\_object\_name();  else  cout<<endl<<path<<" Object is not found";  }  }  return 0;  } |

## **Файл** **application.h**

Листинг 2 – application.h

|  |
| --- |
| #ifndef APPLICATION\_H\_  #define APPLICATION\_H\_  #include <string>  #include "base.h"  #include "cl\_2.h"  #include "cl\_3.h"  #include "cl\_4.h"  #include "cl\_5.h"  #include "cl\_6.h"  class application : public base  {  public:  application(base\* p\_parent);  void bild\_tree\_objects();// метод построения дерева иерархии  int exec\_app();//метод запуска приложения  };  #endif |

## **Файл** **base.cpp**

Листинг 3 – base.cpp

|  |
| --- |
| #include "base.h"  base::base(base\* p\_parent, string object\_name)//конструктор базового класса  {  this->object\_name=object\_name;  this->p\_parent=p\_parent;  state = 0;//отметка неготовности объекта  if (p\_parent)  p\_parent->children.push\_back(this);  }  void base::set\_object\_name(string object\_name)  {  this->object\_name = object\_name;  }  string base::get\_object\_name()  {  return object\_name;  }  void base:: change\_parent (base\* new\_parent)//переопределить головной объект для текущего  {  if (this ->p\_parent==nullptr || new\_parent == nullptr)  return;  for (int i = 0; i<p\_parent ->children.size(); i++)  {  if (p\_parent->children[i] == this)  {  p\_parent->children.erase(p\_parent->children.begin()+i);//удаление текущего объекта из списка наследников объекта-родителя  return ;  }  }  this->p\_parent;  new\_parent->children.push\_back(this);//добавление текущего объекта в список наследников нового головного объекта  }  base\* base :: get\_parent() //получение указателя на головной объект  {return p\_parent;}  int base::get\_state()//получить состояние объекта  {  return state;  }  base\* base::get\_child\_by\_name(string object\_name)  {  if (children.size()==0)  return nullptr;  for( int i = 0; i<children.size(); i++)  {  if (children[i]->get\_object\_name()==object\_name)  return children[i];  }  return nullptr;  }  base\* base :: get\_root()  {  base\* p = this;  while (p->p\_parent)  p = p->p\_parent;  return p;  }  base\* base::get\_object\_by\_name(string name)  {  return get\_root()->find\_object\_by\_name(name);  }  base\* base::find\_object\_by\_name(string name )  {  base\* p;  if (this->object\_name==name) return this;  for (int i = 0; i<children.size();i++)  {  p = children[i]->find\_object\_by\_name(name);  if (p!= nullptr)  return p;  }  return nullptr;  }  void base:: install\_state(int \_state)  {  if (\_state)  {  if (p\_parent==nullptr)  state =\_state;  else if (p\_parent->state)  state =\_state;  }  else if (state)  turn\_off\_state();  }  void base::turn\_off\_state()  {  this -> state = 0;  if (children.size()==0)  return;  for (int i = 0; i<children.size();i++)  {  children[i]->turn\_off\_state();  }  }  void base::print\_object\_tree(base\* parent, int level)  {  string s;  if (level>0)  s.append(4\*level,' ');  if (parent== this)  cout<<parent->get\_object\_name();  else  cout<<endl<<s<<parent->get\_object\_name();  if (parent->children.size()==0)  return;  for (int i = 0; i<parent->children.size();i++)  print\_object\_tree(parent->children[i],level+1);  }  void base::print\_state(base\* parent, int level)  {  string s;  if (level>0) s.append(4\*level,' ');  cout<<endl<<s<<parent->get\_object\_name()<<' ';  if (parent->get\_state())  cout<<"is ready";  else  cout<<"is not ready";  if (parent->children.size()==0)  return;  for (int i =0; i<parent->children.size();i++)  print\_state(parent->children[i],level+1);  }  base\* base::get\_object\_by\_coordinate(string path)  {  if (path == "/")  return get\_root();  if (path == ".")  return this;  if (path.find("//") == 0)  return get\_root()->get\_object\_by\_name(path.substr(2,path.size()-2));  if (path[0] == '/')  return get\_root()->get\_object\_by\_coordinate(path.substr(1,path.size()-1));  int index = path.size();  for (int i = 0; i< path.size(); i++)  {  if (path[i]=='/')  {  index = i;  break;  }  }  string s = path.substr(0,index);  for (int i = 0; i< this->children.size(); i++)  {  if (children[i]->get\_object\_name() == s)  {  if (path.size()==index)  return children[i];  else  return children[i]->get\_object\_by\_coordinate(path.substr(index+1,path.size() - index - 1));  }  }  return nullptr;  }  base::~base()  {  for(int i = 0; i<children.size();i++)  delete children[i];  } |

## **Файл** **base.h**

Листинг 4 – base.h

|  |
| --- |
| #ifndef BASE\_H\_  #define BASE\_H\_  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class base  {  private:  string object\_name;//наименование объекта  base\* p\_parent;//указатель на головной объект  int state; //состояние объекта  vector <base\*>children;//массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии  base\* get\_root();//получение указателя на корневой объект  public:  base( base\* p\_parent , string object\_name = "base" );  void set\_object\_name(string object\_name);  string get\_object\_name();//получить имя объекта  void change\_parent( base\* new\_parent);//переопределить головной объект для текущего  base\* get\_parent();//получить указатель на головной объект для текущего  int get\_state ();//получить состояние объекта  base\* get\_child\_by\_name(string object\_name);//получить указатель на объект потомок по имени объекта  base\* get\_object\_by\_name(string name);//получить указатель на объект по имени  base\* get\_object\_by\_coordinate(string path);  base\* find\_object\_by\_name(string name);//поиск объекта на дереве иерархии по имени  void print\_object\_tree(base\* parent, int level);//вывод дерева иерархии  void install\_state(int \_state);//установка готовности объекта  void turn\_off\_state();//отключение готовности объекта  void print\_state(base\* parent, int level);//вывод дерева иерархии объектов и отметок их готовности  ~base();//деструктор  };  #endif |

## **Файл** **cl\_1.cpp**

Листинг 5 – cl\_1.cpp

|  |
| --- |
| #include "cl\_1.h"  cl\_1::cl\_1(base\* p\_parent, string object\_name):base(p\_parent,object\_name)  {} |

## **Файл** **cl\_1.h**

Листинг 6 – cl\_1.h

|  |
| --- |
| #ifndef CL\_1\_H  #define CL\_1\_H  #include "base.h"  class cl\_1: public base  {  public:  cl\_1(base\* p\_parent, string object\_name);  };  #endif |

## **Файл** **cl\_2.cpp**

Листинг 7 – cl\_2.cpp

|  |
| --- |
| #include "cl\_2.h"  cl\_2::cl\_2(base\* p\_parent, string object\_name):base(p\_parent, object\_name)  {} |

## **Файл** **cl\_2.h**

Листинг 8 – cl\_2.h

|  |
| --- |
| #ifndef CL\_2\_H  #define CL\_2\_H  #include "base.h"  class cl\_2: public base  {  public:  cl\_2(base\* p\_parent, string object\_name);  };  #endif |

## **Файл** **cl\_3.cpp**

Листинг 9 – cl\_3.cpp

|  |
| --- |
| #include "cl\_3.h"  cl\_3::cl\_3(base\* p\_parent, string object\_name):base(p\_parent,object\_name)  {} |

## **Файл** **cl\_3.h**

Листинг 10 – cl\_3.h

|  |
| --- |
| #ifndef CL\_3\_H  #define CL\_3\_H  #include "base.h"  class cl\_3: public base  {  public:  cl\_3(base\* p\_parent, string object\_name);  };  #endif |

## **Файл** **cl\_4.cpp**

Листинг 11 – cl\_4.cpp

|  |
| --- |
| #include "cl\_4.h"  cl\_4::cl\_4(base\* p\_parent, string object\_name):base(p\_parent,object\_name)  {} |

## **Файл** **cl\_4.h**

Листинг 12 – cl\_4.h

|  |
| --- |
| #ifndef CL\_4\_H  #define CL\_4\_H  #include "base.h"  class cl\_4: public base  {  public:  cl\_4(base\* p\_parent, string object\_name);  };  #endif |

## **Файл** **cl\_5.cpp**

Листинг 13 – cl\_5.cpp

|  |
| --- |
| #include "cl\_5.h"  cl\_5::cl\_5(base\* p\_parent, string object\_name):base(p\_parent,object\_name)  {} |

## **Файл** **cl\_5.h**

Листинг 14 – cl\_5.h

|  |
| --- |
| #ifndef CL\_5\_H  #define CL\_5\_H  #include "base.h"  class cl\_5: public base  {  public:  cl\_5(base\* p\_parent, string object\_name);  };  #endif |

## **Файл** **cl\_6.cpp**

Листинг 15 – cl\_6.cpp

|  |
| --- |
| #include "cl\_6.h"  cl\_6::cl\_6(base\* p\_parent, string object\_name):base(p\_parent,object\_name)  {} |

## **Файл** **cl\_6.h**

Листинг 16 – cl\_6.h

|  |
| --- |
| #ifndef CL\_6\_H  #define CL\_6\_H  #include "base.h"  class cl\_6: public base  {  public:  cl\_6(base\* p\_parent, string object\_name);  };  #endif |

## **Файл** **main.cpp**

Листинг 17 – main.cpp

|  |
| --- |
| #include "application.h"  int main()  {  application ob\_application(nullptr);//создание объекта класса application  ob\_application.bild\_tree\_objects();//построение дерева иерархии  return ob\_application.exec\_app();//запуск системы  } |

# 6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 19.

Таблица 19 – 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 |

# ЗАКЛЮЧЕНИЕ

За курс объектно-ориентированного программирования я научился: разрабатывать базовый класс для объектов, определять общий функционал для используемых в рамках приложения объектов, разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева, освоил: алгоритмы обработки структур данных в виде дерева, построение дерева иерархии объектов, переключение состояния объектов или по имени, организацию связей между сигналами и обработчиками объектов, выдачу сигналов от объекта и отработку обработчиков,  реализацию алгоритма решения задачи посредством  последовательной отправки сигналов и отработку  соответствующих обработчиков, реализацию алгоритма решения задачи посредством последовательной отправки сигналов и отработку  соответствующих обработчиков, моделирование движения падающих шариков, описание метода решения программы, написание алгоритма, создание блок-схем. Так-же хочу от всех души поблагодарить своих преподавателей: Путуридзе Зураба Шотовича и Рыжёву Анастасию Андреевну за столь подробное объяснение материала и терпение

# 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 05.05.2021).

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).