Здесь будет титульник, листай ниже

**СОДЕРЖАНИЕ**

[1 ПОСТАНОВКА ЗАДАЧИ 6](#__RefHeading___Toc8050_4093944657)

[1.1 Описание входных данных 8](#__RefHeading___Toc8052_4093944657)

[1.2 Описание выходных данных 9](#__RefHeading___Toc8054_4093944657)

[2 МЕТОД РЕШЕНИЯ 10](#__RefHeading___Toc8056_4093944657)

[3 ОПИСАНИЕ АЛГОРИТМОВ 15](#__RefHeading___Toc8058_4093944657)

[3.1 Алгоритм метода build\_tree\_objects класса cl\_application 15](#__RefHeading___Toc8060_4093944657)

[3.2 Алгоритм метода exec\_app класса cl\_application 17](#__RefHeading___Toc8062_4093944657)

[3.3 Алгоритм метода ObjNameCount класса cl\_base 17](#__RefHeading___Toc8064_4093944657)

[3.4 Алгоритм метода SearchObjOnBranch класса cl\_base 18](#__RefHeading___Toc8066_4093944657)

[3.5 Алгоритм метода SearchObjOnTree класса cl\_base 19](#__RefHeading___Toc8068_4093944657)

[3.6 Алгоритм метода PrintObjects класса cl\_base 20](#__RefHeading___Toc8070_4093944657)

[3.7 Алгоритм метода PrintObjectsStates класса cl\_base 21](#__RefHeading___Toc8072_4093944657)

[3.8 Алгоритм метода SetObjectState класса cl\_base 22](#__RefHeading___Toc8074_4093944657)

[3.9 Алгоритм метода GetObjectState класса cl\_base 22](#__RefHeading___Toc8076_4093944657)

[3.10 Алгоритм функции main 23](#__RefHeading___Toc8078_4093944657)

[3.11 Алгоритм конструктора класса cl\_2 23](#__RefHeading___Toc8080_4093944657)

[3.12 Алгоритм конструктора класса cl\_3 24](#__RefHeading___Toc8082_4093944657)

[3.13 Алгоритм конструктора класса cl\_4 24](#__RefHeading___Toc8084_4093944657)

[3.14 Алгоритм конструктора класса cl\_5 24](#__RefHeading___Toc8086_4093944657)

[3.15 Алгоритм конструктора класса cl\_6 25](#__RefHeading___Toc8088_4093944657)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 26](#__RefHeading___Toc8090_4093944657)

[5 КОД ПРОГРАММЫ 37](#__RefHeading___Toc8092_4093944657)

[5.1 Файл cl\_1.cpp 37](#__RefHeading___Toc8094_4093944657)

[5.2 Файл cl\_1.h 37](#__RefHeading___Toc8096_4093944657)

[5.3 Файл cl\_2.cpp 37](#__RefHeading___Toc8098_4093944657)

[5.4 Файл cl\_2.h 38](#__RefHeading___Toc8100_4093944657)

[5.5 Файл cl\_3.cpp 38](#__RefHeading___Toc8102_4093944657)

[5.6 Файл cl\_3.h 38](#__RefHeading___Toc8104_4093944657)

[5.7 Файл cl\_4.cpp 38](#__RefHeading___Toc8106_4093944657)

[5.8 Файл cl\_4.h 39](#__RefHeading___Toc8108_4093944657)

[5.9 Файл cl\_5.cpp 39](#__RefHeading___Toc8110_4093944657)

[5.10 Файл cl\_5.h 39](#__RefHeading___Toc8112_4093944657)

[5.11 Файл cl\_6.cpp 39](#__RefHeading___Toc8114_4093944657)

[5.12 Файл cl\_6.h 40](#__RefHeading___Toc8116_4093944657)

[5.13 Файл cl\_application.cpp 40](#__RefHeading___Toc8118_4093944657)

[5.14 Файл cl\_application.h 41](#__RefHeading___Toc8120_4093944657)

[5.15 Файл cl\_base.cpp 42](#__RefHeading___Toc8122_4093944657)

[5.16 Файл cl\_base.h 45](#__RefHeading___Toc8124_4093944657)

[5.17 Файл main.cpp 45](#__RefHeading___Toc8126_4093944657)

[6 ТЕСТИРОВАНИЕ 47](#__RefHeading___Toc8128_4093944657)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#__RefHeading___Toc8130_4093944657)

# 1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

**Вывод иерархического дерева объектов на консоль.**

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:

* метод поиска объекта на ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
* метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
* метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
* метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
* метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется. При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

* Построить дерево объектов системы (в методе корневого объекта построения исходного дерева объектов).
* В методе корневого объекта запуска моделируемой системы реализовать:
  + Вывод на консоль иерархического дерева объектов в следующем виде:

root

ob\_1

ob\_2

ob\_3

ob\_4

ob\_5

ob\_6

ob\_7​

где: root - наименование корневого объекта (приложения).

* + Переключение готовности объектов согласно входным данным (командам).
  + Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

root is ready

ob\_1 is ready

ob\_2 is ready

ob\_3 is ready

ob\_4 is not ready

ob\_5 is not ready

ob\_6 is ready

ob\_7 is not ready​

## 1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

**Первая строка:**

«Наименование корневого объекта»

**Со второй строки:**

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

. . . . .

endtree​

Со следующей строки вводятся команды включения или отключения объектов

«Наименование объекта» «Номер состояния объекта»

**Пример ввода:**

app\_root

app\_root object\_01 3

app\_root object\_02 2

object\_02 object\_04 3

object\_02 object\_05 5

object\_01 object\_07 2

endtree

app\_root 1

object\_07 3

object\_01 1

object\_02 -2

object\_04 1

## 1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

Object tree

«Наименование корневого объекта»

«Наименование объекта 1»

«Наименование объекта 2»

«Наименование объекта 3»

. . . . .

The tree of objects and their readiness

«Наименование корневого объекта» «Отметка готовности»

«Наименование объекта 1» «Отметка готовности»

«Наименование объекта 2» «Отметка готовности»

«Наименование объекта 3» «Отметка готовности»

. . . . .

«Отметка готовности» - равно «is ready» или «is not ready»

Отступ каждого уровня иерархии 4 позиции.

**Пример вывода:**

Object tree

app\_root

object\_01

object\_07

object\_02

object\_04

object\_05

The tree of objects and their readiness

app\_root is ready

object\_01 is ready

object\_07 is not ready

object\_02 is ready

object\_04 is ready

object\_05 is not ready

# 2 МЕТОД РЕШЕНИЯ

Для решения задачи используются то же, что и в работе КВ1, однако было добавлено и изменено несколько классов и методов.

* Таблица 1 – Иерархия наследования классов

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | | Имя класса | Классы-наследники | | Модификатор доступа при наследовании | Описание | Номер | Комментарий |
| 1 | | cl\_base |  | |  | Базовый класс в иерархии классов. Содержит основные поля и методы |  |  |
|  | | cl\_application | public | |  | 2 |  |
|  | | cl\_1 | public | |  | 3 |  |
|  | | cl\_2 | public | |  | 4 |  |
|  | | cl\_3 | public | |  | 5 |  |
|  | | cl\_4 | public | |  | 6 |  |
|  | | cl\_5 | public | |  | 7 |  |
|  | | cl\_6 | public | |  | 8 |  |
| 2 | | cl\_application |  | |  | Класс корневого объекта дерева иерархии |  |  |
| 3 | | cl\_1 |  | |  | Класс объекта дерева иерархии |  |  |
| 4 | | cl\_2 |  | |  | Класс объекта дерева иерархии |  |  |
| 5 | | cl\_3 |  | |  | Класс объекта дерева иерархии |  |  |
| 6 | | cl\_4 |  | |  | Класс объекта дерева иерархии |  |  |
| 7 | | cl\_5 |  | |  | Класс объекта дерева иерархии |  |  |
| 8 | | cl\_6 |  | |  | Класс объекта дерева иерархии |  |  |

* Класс cl\_base:
  + Свойства/поля:
    - Те же поля как в КВ1 у класса cl\_base
    - Поле, отвечающее за хранение состояния объекта:
      * Наименование - state
      * Тип - bool
      * Модификатор доступа - private
  + Методы:
    - Те же методы как в КВ1 у класса cl\_base
    - Метод ObjNameCount
      * Функционал - Используется для счета объектов в иерархии с определенным наименованием
    - Метод SearchObjBranch
      * Функционал - Используется для поиска объекта на ветке дерева иерархии от текущего по имени
    - Метод SearchObjOnTree
      * Функционал - Используется для поиска объекта на дереве иерархии по имени
    - Метод PrintObjects
      * Функционал - Используется для вывода иерархии объектов (дерева или ветки) и отметок их состояний от текущего объекта
    - Метод SetObjectState
      * Функционал - Используется для установки состояния объекта
    - Метод GetObjectState
      * Функционал - Используется для возвращения состояние объекта
* Класс cl\_2:
  + Методы:
    - Конструктор cl\_2
      * Функционал - Параметризированный конструктором с параметром указателя на головной объект создаваемого объекта и его наименованием
* Класс cl\_3:
  + Методы:
    - Конструктор cl\_3
      * Функционал - Параметризированный конструктором с параметром указателя на головной объект создаваемого объекта и его наименованием
* Класс cl\_4:
  + Методы:
    - Конструктор cl\_3
      * Функционал - Параметризированный конструктором с параметром указателя на головной объект создаваемого объекта и его наименованием;
* Класс cl\_4:
  + Методы:
    - Конструктор cl\_4
      * Функционал - Параметризированный конструктором с параметром указателя на головной объект создаваемого объекта и его наименованием;
* Класс cl\_5:
  + Методы:
    - Конструктор cl\_5
      * Функционал - Параметризированный конструктором с параметром указателя на головной объект создаваемого объекта и его наименованием;
* Класс cl\_6:
  + Методы:
    - Конструктор cl\_6
      * Функционал - Параметризированный конструктором с параметром указателя на головной объект создаваемого объекта и его наименованием;

# 3 ОПИСАНИЕ АЛГОРИТМОВ

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

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

Функционал: Используется для построения 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 headName и childName | 2 |
| 2 |  | Объявление целочисленных переменных classNum и state | 3 |
| 3 |  | Ввод значения переменной headName | 4 |
| 4 |  | Вызов метода SetName текущего объекта с аргументом значения headName | 5 |
| 5 |  | Ввод значения переменной headName | 6 |
| 6 | Значение headName равно "endtree" | Выход из цикла | 15 |
|  | Инициализация переменной-объекта parentObj типа указателя на cl\_base, путём вызова метода  SearchObjOnTree с аргументом значения headName | 7 |
| 7 |  | Ввод значений переменных childName и classNum | 8 |
| 8 |  | Вызов метода getChild с входным параметром childName для parentObj | 9 |
| 9 | parentObj не равен nullptr и результат метода getChild равен nullptr |  | 10 |
|  |  | 6 |
| 10 | classNum равно 2 | Создание объекта типа cl\_2 с аргументами в виде значений parentObj и childName | 6 |
|  |  | 11 |
| 11 | classNum равно 3 | Создание объекта типа cl\_3 с аргументами в виде значений parentObj и childName | 6 |
|  |  | 12 |
| 12 | classNum равно 4 | Создание объекта типа cl\_4 с аргументами в виде значений parentObj и childName | 6 |
|  |  | 13 |
| 13 | classNum равно 5 | Создание объекта типа cl\_5 с аргументами в виде значений parentObj и childName | 6 |
|  |  | 14 |
| 14 | classNum равно 6 | Создание объекта типа cl\_6 с аргументами в виде значений parentObj и childName | 6 |
|  |  | 15 |
| 15 | Продолжается поток входных данных | Ввод значений переменных headName и state | 16 |
|  |  | ∅ |
| 16 |  | Инициализация переменной-объекта obj универсального типа, путём вызова метода SearchObjOnTree с аргументом значения headName | 17 |
| 17 | obj не равен nullptr | Вызов метода SetObjectState переменной-объекта obj с аргументом значения state | 15 |
|  |  | 15 |

## Алгоритм метода exec\_app класса cl\_application

Функционал: Используется для запуска приложения (начало функционирования системы).

Параметры: Отсутствуют.

Возвращаемое значение: bool - корректность выполнения метода.

Алгоритм метода представлен в таблице 3.

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

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод "Object tree" на экран | 2 |
| 2 |  | Вызов метода PrintObjects текущего объекта | 3 |
| 3 |  | Вывод "\nThe tree of objects and their readiness" на экран | 4 |
| 4 |  | Вызов метода PrintObjectsStates текущего объекта | 5 |
| 5 |  | Возвращение значения 0 | ∅ |

## Алгоритм метода ObjNameCount класса cl\_base

Функционал: Счет объектов в иерархии с наименованием в виде значения параметра objName.

Параметры: string objName - имя для поиска объектов и их подсчет.

Возвращаемое значение: int - количество объектов с наименованием objName.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода ObjNameCount класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация целочисленной переменной count значением 0 | 2 |
| 2 | Значение наименования текущего объекта равно objName | Инкремент значения переменной count | 3 |
|  |  | 3 |
| 3 |  | Объявление переменной-объекта child универсального типа | 4 |
| 4 | child находится в списке children | Прибавление к переменной count значение выполнения метода ObjNameCount, с аргументом в виде значения objName, переменной-объекта child | 4 |
|  | Возврат значения переменной count | ∅ |

## Алгоритм метода SearchObjOnBranch класса cl\_base

Функционал: Поиск объекта на ветке дерева иерархии от текущего по имени.

Параметры: string objName - имя для поиска объекта.

Возвращаемое значение: cl\_base\* - указатель на объект, найденный на ветке дерева.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода SearchObjOnBranch класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Значение наименования текущего объекта равно objName | Возврат указателя на текущий объект | ∅ |
|  |  | 2 |
| 2 |  | Объявление переменной-объекта child универсального типа | 3 |
| 3 | child находится в списке children | Инициализация переменной-объекта subChild типа указателя на cl\_base, путём вызова метода SearchObjOnBranch с аргументом значения objName | 4 |
|  | Возврат значения nullptr | ∅ |
| 4 | subChild не равен nullptr | Возврат указателя на subChild | ∅ |
|  |  | 3 |

## Алгоритм метода SearchObjOnTree класса cl\_base

Функционал: Поиск объекта на дереве иерархии от корневого по имени.

Параметры: string objName - имя для поиска объекта.

Возвращаемое значение: cl\_base\* - указатель на объект, найденный на дереве.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода SearchObjOnTree класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Инициализация переменной-объекта obj универсального типа указателем на текущий объект | 2 |
| 2 | Obj имеет родительский объект | Присваивание obj значение родительского объекта obj | 2 |
|  |  | 3 |
| 3 |  | Вызов метода ObjNameCount с аргументом objName | 4 |
| 4 | Возвращаемое значение ObjNameCount не равно 1 | Возврат значения nullptr | ∅ |
|  | Возврат значения выполнения метода SearchObjOnBranch с аргументом objName | ∅ |

## Алгоритм метода PrintObjects класса cl\_base

Функционал: Вывод иерархии объектов (дерева или ветки) от текущего объекта.

Параметры: int spaces - длина отступа от края.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода PrintObjects класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод наименования текущего объекта | 2 |
| 2 | Список children текущего объекта не пустой | Объявление переменной-объекта child универсального типа | 3 |
|  |  | ∅ |
| 3 | child находится в списке children | Переход на новую строку | 4 |
|  |  | ∅ |
| 4 |  | Инициализация целочисленной переменной i значением 0 | 5 |
| 5 | Значение i меньше значения spaces | Вывод пробела | 6 |
|  |  | 7 |
| 6 |  | Инкремент значения i | 7 |
| 7 |  | Вызов метода PrintObjects, с аргументом spaces+4, переменной-объекта child | 3 |

## Алгоритм метода PrintObjectsStates класса cl\_base

Функционал: Вывод иерархии объектов (дерева или ветки) и отметок их состояний от текущего объекта.

Параметры: int spaces - длина отступа от края.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода PrintObjectsStates класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод наименования текущего объекта | 2 |
| 2 |  | Вызов метода GetObjectState | 3 |
| 3 | Значение выполнения метода GetObjectState истинно | Вывод " is ready" | 4 |
|  | Вывод " is not ready" | 4 |
| 4 | Список children текущего объекта не пустой | Объявление переменной-объекта child универсального типа | 5 |
|  |  | ∅ |
| 5 | child находится в списке children | Переход на новую строку | 6 |
|  |  | ∅ |
| 6 |  | Инициализация целочисленной переменной i значением 0 | 7 |
| 7 | Значение i меньше значения spaces | Вывод пробела | 8 |
|  |  | 9 |
| 8 |  | Инкремент значения i | 7 |
| 9 |  | Вызов метода PrintObjectsStates, с аргументом spaces+4, переменной-объекта child | 5 |

## Алгоритм метода SetObjectState класса cl\_base

Функционал: Установка состояния объекта.

Параметры: bool state - значение состояния для установки текущему объекту.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода SetObjectState класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Текущий объект имеет родительский объект, который имеет выключенное состояние | Присваивание полю state текущего объекта значение false | 2 |
|  | Присваивание полю state текущего объекта значение параметра state | 2 |
| 2 | Параметр state имеет значение false | Объявление переменной-объекта child универсального типа | 3 |
|  |  | ∅ |
| 3 | child находится в списке children | Вызов метода SetObjectState, с аргументом значения параметра state, переменной-объекта child | ∅ |
|  |  | ∅ |

## Алгоритм метода GetObjectState класса cl\_base

Функционал: Возврат значения состояния текущего объекта.

Параметры: Отсутствуют.

Возвращаемое значение: bool - значение состояния текущего объекта.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода GetObjectState класса cl\_base

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

## Алгоритм функции main

Функционал: Основной алгоритм работы программы.

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения программы.

Алгоритм функции представлен в таблице 11.

Таблица 11 – Алгоритм функции main

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Выполнения алгоритма данной функции из предыдущей работы (КВ1) | ∅ |

## Алгоритм конструктора класса cl\_2

Функционал: Создание объекта на основе класса cl\_2.

Параметры: cl\_base\* parent - указатель на родительский объект создаваемого объекта, string name - наименование создаваемого объекта.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl\_2

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с аргументами parent и name, являющимися параметрами конструктора текущего класса | ∅ |

## Алгоритм конструктора класса cl\_3

Функционал: Создание объекта на основе класса cl\_3.

Параметры: cl\_base\* parent - указатель на родительский объект создаваемого объекта, string name - наименование создаваемого объекта.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса cl\_3

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с аргументами parent и name, являющимися параметрами конструктора текущего класса | ∅ |

## Алгоритм конструктора класса cl\_4

Функционал: Создание объекта на основе класса cl\_4.

Параметры: cl\_base\* parent - указатель на родительский объект создаваемого объекта, string name - наименование создаваемого объекта.

Алгоритм конструктора представлен в таблице 14.

Таблица 14 – Алгоритм конструктора класса cl\_4

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с аргументами parent и name, являющимися параметрами конструктора текущего класса | ∅ |

## Алгоритм конструктора класса cl\_5

Функционал: Создание объекта на основе класса cl\_5.

Параметры: cl\_base\* parent - указатель на родительский объект создаваемого объекта, string name - наименование создаваемого объекта.

Алгоритм конструктора представлен в таблице 15.

Таблица 15 – Алгоритм конструктора класса cl\_5

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с аргументами parent и name, являющимися параметрами конструктора текущего класса | ∅ |

## Алгоритм конструктора класса cl\_6

Функционал: Создание объекта на основе класса cl\_6.

Параметры: cl\_base\* parent - указатель на родительский объект создаваемого объекта, string name - наименование создаваемого объекта.

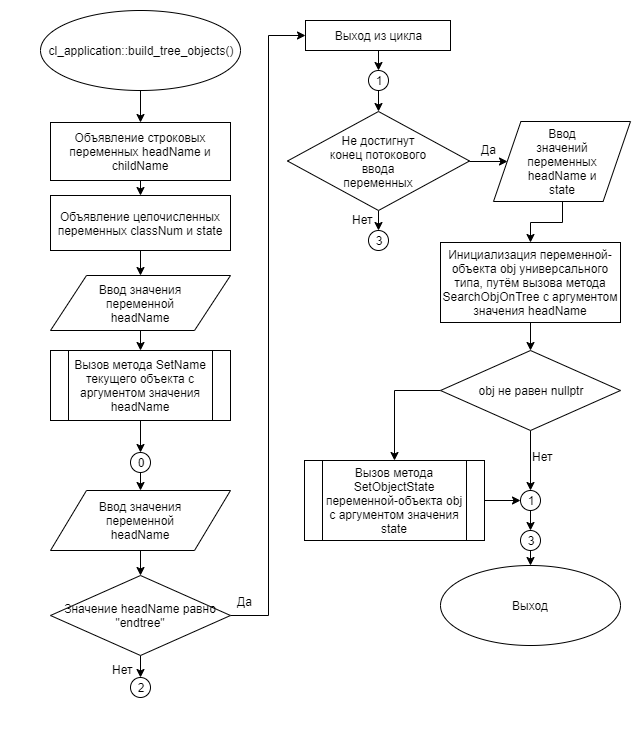
Алгоритм конструктора представлен в таблице 16.

Таблица 16 – Алгоритм конструктора класса cl\_6

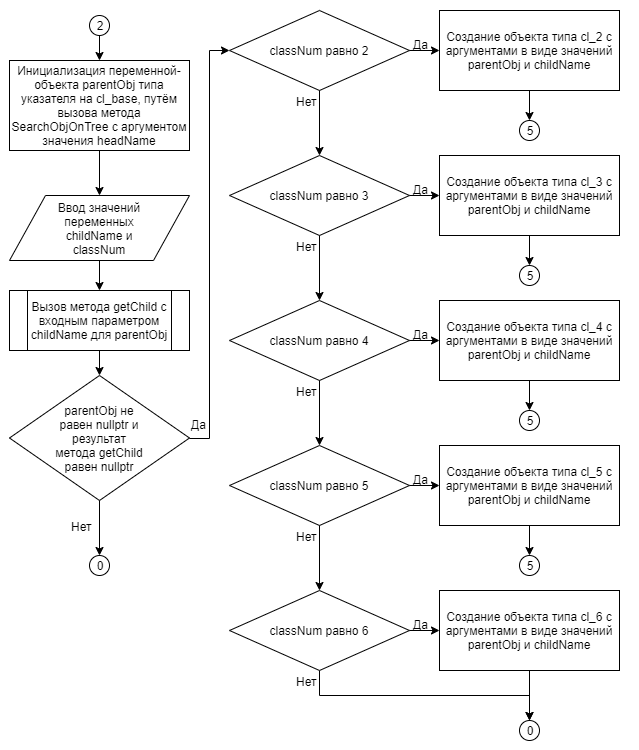
| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с аргументами parent и name, являющимися параметрами конструктора текущего класса | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

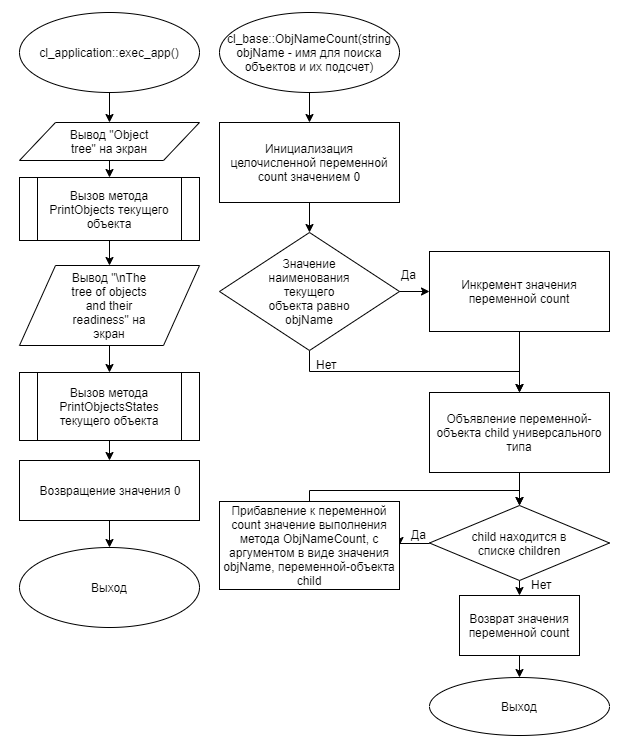
Представим описание алгоритмов в графическом виде на рисунках 1-11.



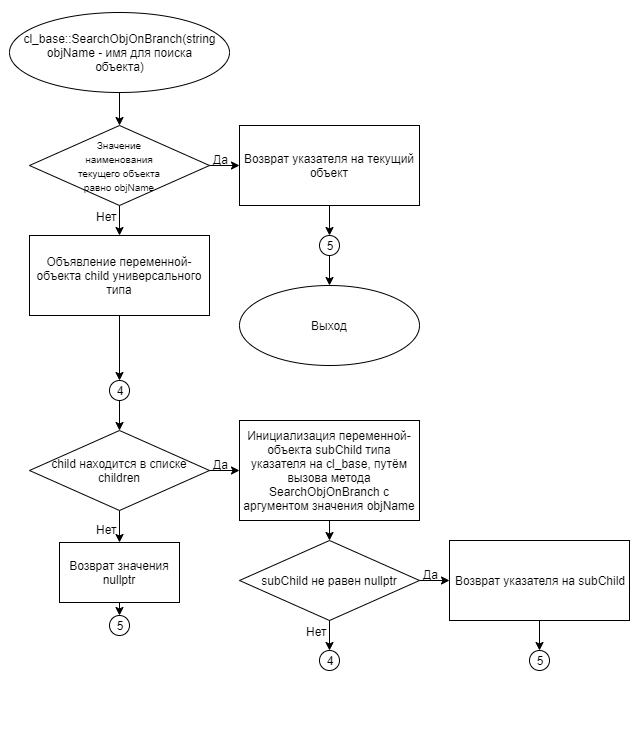
**Рисунок 1 – Блок-схема алгоритма**



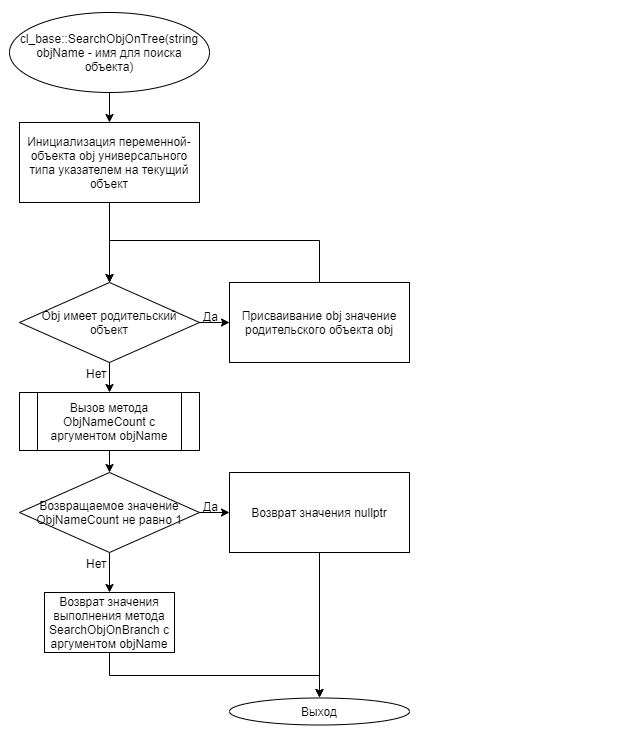
**Рисунок 2 – Блок-схема алгоритма**



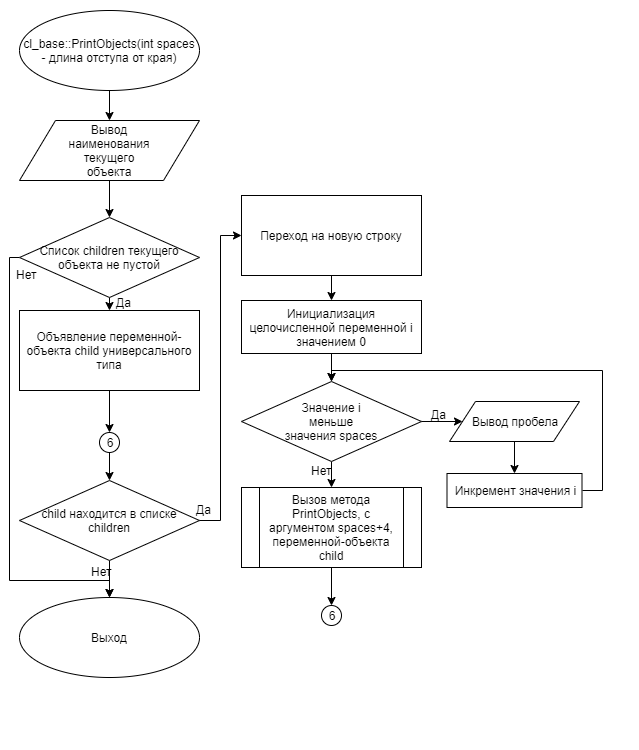
**Рисунок 3 – Блок-схема алгоритма**



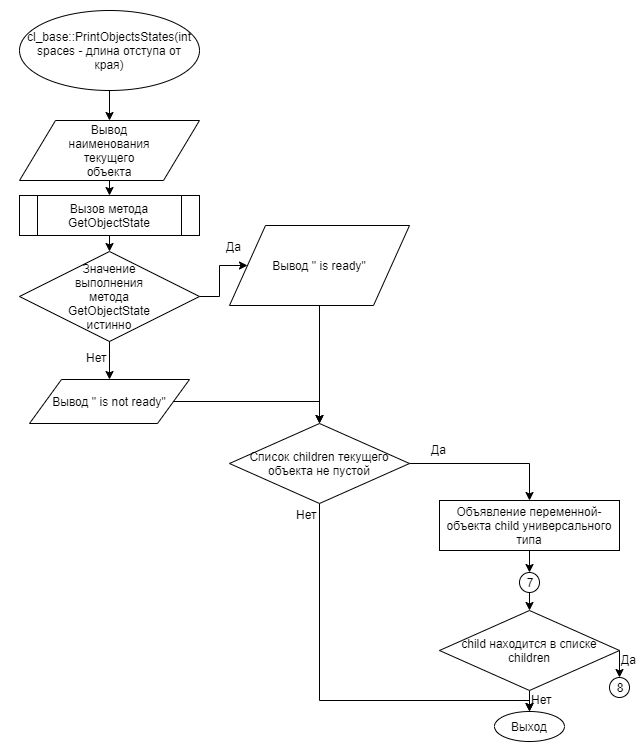
**Рисунок 4 – Блок-схема алгоритма**



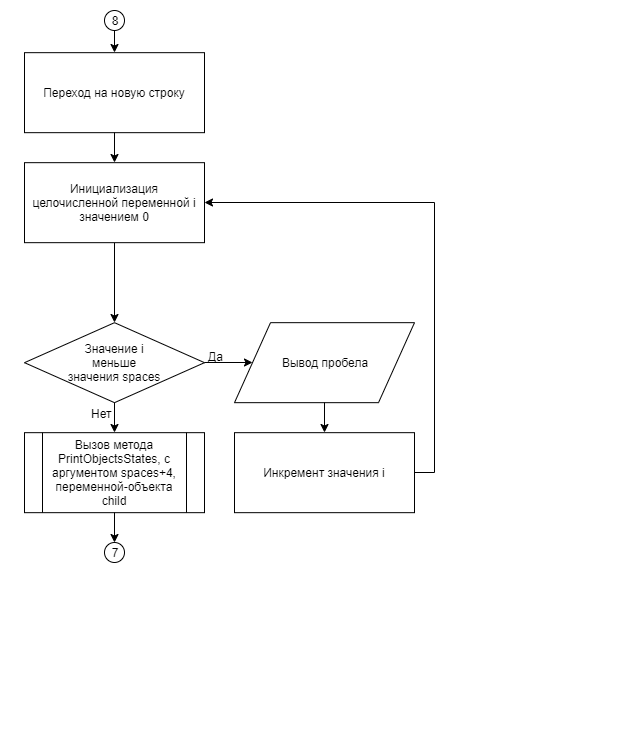
**Рисунок 5 – Блок-схема алгоритма**



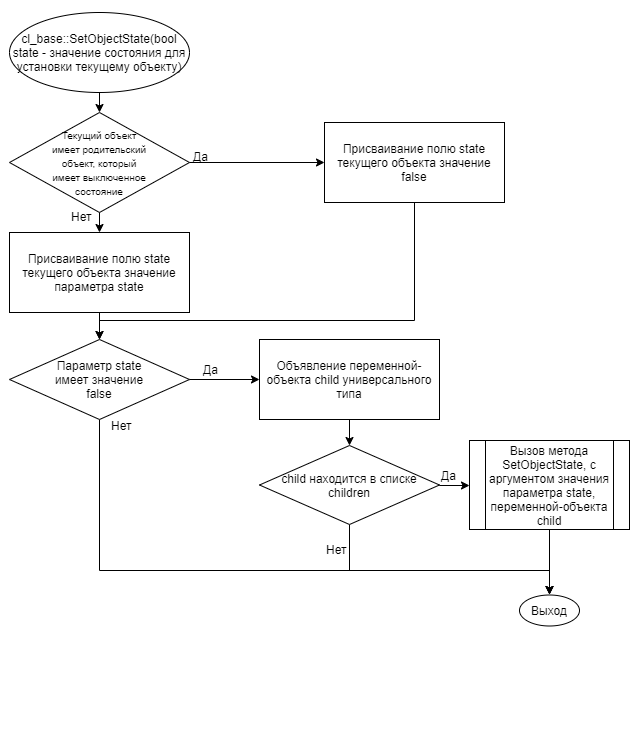
**Рисунок 6 – Блок-схема алгоритма**



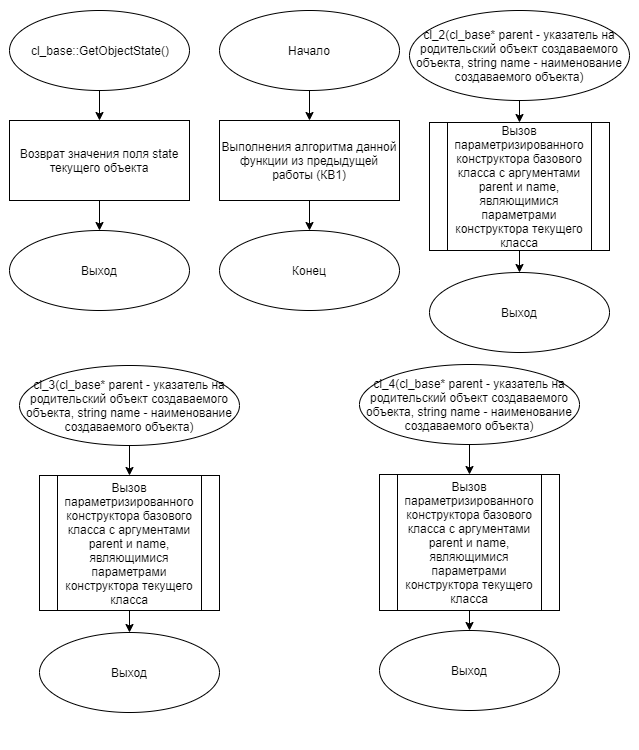
**Рисунок 7 – Блок-схема алгоритма**



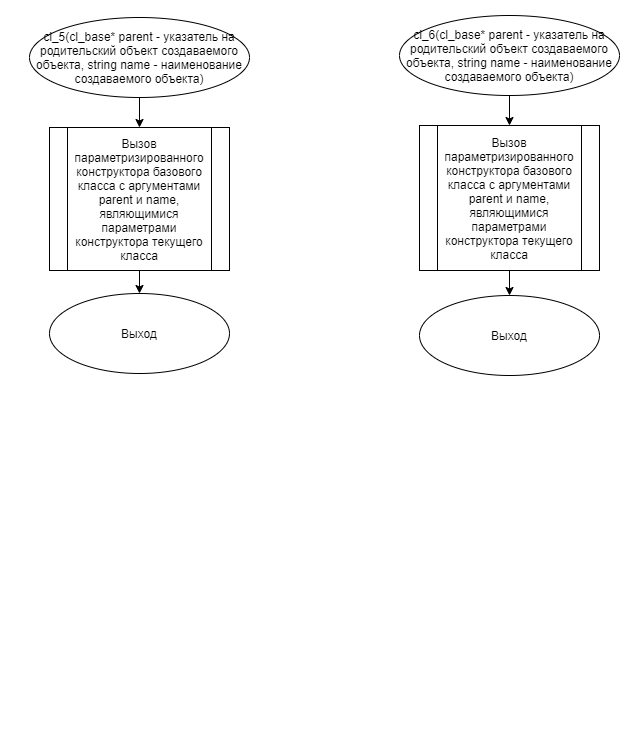
**Рисунок 8 – Блок-схема алгоритма**



**Рисунок 9 – Блок-схема алгоритма**



**Рисунок 10 – Блок-схема алгоритма**



**Рисунок 11 – Блок-схема алгоритма**

# 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

## **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"  // Конструктор класса cl\_application, вызывает конструктор базового класса cl\_base  cl\_application::cl\_application(cl\_base\* parent) : cl\_base(parent)  {}  // Метод для построения дерева объектов  void cl\_application::build\_tree\_objects()  {  string headName, childName;  int classNum, state;  // Считываем имя корневого объекта  cin >> headName;  SetName(headName);  // Цикл для добавления объектов в дерево  while(true)  {  // Считываем имя родительского объекта  cin >> headName;  // Если встретили ключевое слово "endtree", то выходим из цикла  if (headName == "endtree")  {  break;  }  // Ищем родительский объект на дереве  cl\_base\* parentObj = SearchObjOnTree(headName);  // Считываем имя дочернего объекта и номер его класса  cin >> childName >> classNum;  // Если родительский объект найден и у него нет такого дочернего объекта, то добавляем его на дерево  if (parentObj && !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;  }  }  }  }  // Цикл для установки состояний объектов  while(cin >> headName >> state)  {  auto obj = SearchObjOnTree(headName);  if(obj)  {  obj->SetObjectState(state);  }  }  }  // Метод для запуска приложения  int cl\_application::exec\_app()  {  // Выводим дерево объектов и их готовности  cout << "Object tree" << endl;  PrintObjects();  cout << "\nThe tree of objects and their readiness" << endl;  PrintObjectsStates();  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)  {  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  {  for (auto child : children)  {  if (child->GetName() == objName)  {  return child;  }  }  return nullptr;  }  int cl\_base::ObjNameCount(string objName)  {  int count = 0;  if(GetName() == objName)  {  count++;  }  for (auto child : children)  {  count += child->ObjNameCount(objName);  }  return count;  }  cl\_base\* cl\_base::SearchObjOnBranch(string objName)  {  if (GetName() == objName)  {  return this;  }  for (auto child : children)  {  cl\_base\* subChild = child->SearchObjOnBranch(objName);  if (subChild)  {  return subChild;  }  }  return nullptr;  }  cl\_base\* cl\_base::SearchObjOnTree(string objName)  {  auto obj = this;  while(obj->GetParent())  {  obj = obj->GetParent();  }  if (ObjNameCount(objName) != 1)  {  return nullptr;  }  return SearchObjOnBranch(objName);  }  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);  }  }  }  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;  } |

## **Файл** **cl\_base.h**

Листинг 16 – cl\_base.h

|  |
| --- |
| #ifndef CL\_BASE\_H  #define CL\_BASE\_H  #include <iostream>  #include <string>  #include <vector>  using namespace std;  // Объявление класса cl\_base  class cl\_base  {  string name; // Имя объекта  cl\_base\* parent; // Указатель на родительский объект  bool state = false; // Состояние объекта (вкл/выкл)  vector<cl\_base\*> children; // Вектор дочерних объектов  public:  cl\_base(cl\_base\* parent, string name = "Base\_object"); // Конструктор класса  ~cl\_base(); // Деструктор класса  bool SetName(string newName); // Метод для изменения имени объекта  string GetName() const; // Метод для получения имени объекта  cl\_base\* GetParent() const; // Метод для получения указателя на родительский объект  cl\_base\* GetChild(string objName) const; // Метод для получения указателя на дочерний объект по имени  int ObjNameCount(string objName); // Метод для подсчета количества объектов с заданным именем на ветке  cl\_base\* SearchObjOnBranch(string objName); // Метод для поиска объекта с заданным именем на ветке  cl\_base\* SearchObjOnTree(string objName); // Метод для поиска объекта с заданным именем на всем дереве объектов  void PrintObjects(int spaces = 4) const; // Метод для вывода всех объектов на экран  void PrintObjectsStates(int spaces = 4) const; // Метод для вывода состояний всех объектов на экран  void SetObjectState(bool state); // Метод для изменения состояния объекта  bool GetObjectState() const; // Метод для получения состояния объекта  };  #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 17.

Таблица 17 – Результат тестирования программы

| Входные данные | Ожидаемые выходные данные | Фактические выходные данные |
| --- | --- | --- |
| app\_root  app\_root object\_01 3  app\_root object\_02 2  object\_02 object\_04 3  object\_02 object\_05 5  object\_01 object\_07 2  endtree  app\_root 1  object\_07 3  object\_01 1  object\_02 -2  object\_04 1  app\_root 0 | Object tree  app\_root  object\_01  object\_07  object\_02  object\_04  object\_05  The tree of objects and their readiness  app\_root is not ready  object\_01 is not ready  object\_07 is not ready  object\_02 is not ready  object\_04 is not ready  object\_05 is not ready | Object tree  app\_root  object\_01  object\_07  object\_02  object\_04  object\_05  The tree of objects and their readiness  app\_root is not ready  object\_01 is not ready  object\_07 is not ready  object\_02 is not ready  object\_04 is not ready  object\_05 is not ready |
| app\_root  app\_root object\_01 3  app\_root object\_02 2  app\_root object\_02 6  object\_02 object\_04 3  object\_02 object\_05 5  object\_01 object\_07 2  endtree  app\_root 1  object\_07 3  object\_01 1  object\_02 -2  object\_04 1 | Object tree  app\_root  object\_01  object\_07  object\_02  object\_04  object\_05  The tree of objects and their readiness  app\_root is ready  object\_01 is ready  object\_07 is not ready  object\_02 is ready  object\_04 is ready  object\_05 is not ready | Object tree  app\_root  object\_01  object\_07  object\_02  object\_04  object\_05  The tree of objects and their readiness  app\_root is ready  object\_01 is ready  object\_07 is not ready  object\_02 is ready  object\_04 is ready  object\_05 is not ready |
| app\_root  app\_root object\_01 3  app\_root object\_02 2  object\_02 object\_04 3  object\_02 object\_05 5  object\_01 object\_07 2  endtree  app\_root 1  object\_07 3  object\_01 1  object\_02 -2  object\_04 1 | Object tree  app\_root  object\_01  object\_07  object\_02  object\_04  object\_05  The tree of objects and their readiness  app\_root is ready  object\_01 is ready  object\_07 is not ready  object\_02 is ready  object\_04 is ready  object\_05 is not ready | Object tree  app\_root  object\_01  object\_07  object\_02  object\_04  object\_05  The tree of objects and their readiness  app\_root is ready  object\_01 is ready  object\_07 is not ready  object\_02 is ready  object\_04 is ready  object\_05 is not ready |

# 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).