Здесь будет титульник, листай ниже

**СОДЕРЖАНИЕ**

[1 ПОСТАНОВКА ЗАДАЧИ 6](#__RefHeading___Toc5581_128800910)

[1.1 Описание входных данных 8](#__RefHeading___Toc5583_128800910)

[1.2 Описание выходных данных 8](#__RefHeading___Toc5585_128800910)

[2 МЕТОД РЕШЕНИЯ 10](#__RefHeading___Toc5587_128800910)

[3 ОПИСАНИЕ АЛГОРИТМОВ 14](#__RefHeading___Toc5589_128800910)

[3.1 Алгоритм конструктора класса cl\_base 14](#__RefHeading___Toc5591_128800910)

[3.2 Алгоритм деструктора класса cl\_base 14](#__RefHeading___Toc5593_128800910)

[3.3 Алгоритм метода SetName класса cl\_base 15](#__RefHeading___Toc5595_128800910)

[3.4 Алгоритм метода GetName класса cl\_base 16](#__RefHeading___Toc5597_128800910)

[3.5 Алгоритм метода GetParent класса cl\_base 16](#__RefHeading___Toc5599_128800910)

[3.6 Алгоритм метода PrintObjectTree класса cl\_base 16](#__RefHeading___Toc5601_128800910)

[3.7 Алгоритм метода GetChild класса cl\_base 17](#__RefHeading___Toc5603_128800910)

[3.8 Алгоритм конструктора класса cl\_application 18](#__RefHeading___Toc5605_128800910)

[3.9 Алгоритм метода build\_tree\_objects класса cl\_application 19](#__RefHeading___Toc5607_128800910)

[3.10 Алгоритм метода exec\_app класса cl\_application 20](#__RefHeading___Toc5609_128800910)

[3.11 Алгоритм конструктора класса cl\_1 20](#__RefHeading___Toc5611_128800910)

[3.12 Алгоритм функции main 21](#__RefHeading___Toc5613_128800910)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 22](#__RefHeading___Toc5615_128800910)

[5 КОД ПРОГРАММЫ 33](#__RefHeading___Toc5617_128800910)

[5.1 Файл cl\_1.cpp 33](#__RefHeading___Toc5619_128800910)

[5.2 Файл cl\_1.h 33](#__RefHeading___Toc5621_128800910)

[5.3 Файл cl\_application.cpp 33](#__RefHeading___Toc5623_128800910)

[5.4 Файл cl\_application.h 34](#__RefHeading___Toc5625_128800910)

[5.5 Файл cl\_base.cpp 35](#__RefHeading___Toc5627_128800910)

[5.6 Файл cl\_base.h 36](#__RefHeading___Toc5629_128800910)

[5.7 Файл main.cpp 37](#__RefHeading___Toc5631_128800910)

[6 ТЕСТИРОВАНИЕ 38](#__RefHeading___Toc5633_128800910)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#__RefHeading___Toc5635_128800910)

# 1 ПОСТАНОВКА ЗАДАЧИ

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов. В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Каждый объект на дереве иерархии имеет свое место и наименование. Не допускается для одного головного объекта одинаковые наименования в составе подчиненных объектов.

Создать базовый класс со следующими элементами:

* Свойства:
  + Наименование объекта (строкового типа);
  + Указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно nullptr);
  + Динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.
* Функционал:
  + Параметризированный конструктор с параметрами: указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии; строкового типа, содержащий наименование создаваемого объекта (имеет значение по умолчанию);
  + Метод редактирования имени объекта. Один параметр строкового типа, содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
  + Метод получения имени объекта;
  + Метод получения указателя на головной объект текущего объекта;
  + Метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
  + Метод получения указателя на непосредственно подчиненный объект по его имени. Если объект не найден, то возвращает nullptr. Один параметр строкового типа, содержит наименование искомого подчиненного объекта.

Для построения дерева иерархии объектов в качестве корневого объекта используется объект приложение. Класс объекта приложения наследуется от базового класса. Объект приложение реализует следующий функционал:

* Метод построения исходного дерева иерархии объектов (конструирования моделируемой системы);
* Метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз. Переход на новый уровень происходит только от правого (последнего) объекта предыдущего уровня. Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Исключить создание объекта если его наименование совпадает с именем уже имеющегося подчиненного объекта у предполагаемого головного. Исключить добавление нового объекта, не последнему подчиненному предыдущего уровня.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

int main ( )

{

cl\_application ob\_cl\_application ( nullptr ); // создание корневого объекта

ob\_cl\_application.build\_tree\_objects ( ); // конструирование системы, построение дерева объектов

return ob\_cl\_application.exec\_app ( ); // запуск системы

}

Наименование класса cl\_application и идентификатора корневого объекта ob\_cl\_application могут быть изменены разработчиком.

Все версии курсовой работы имеют такую основную функцию.

## 1.1 Описание входных данных

**Первая строка:**

«имя корневого объекта»

**Вторая строка и последующие строки:**

«имя головного объекта» «имя подчиненного объекта»

Создается подчиненный объект и добавляется в иерархическое дерево. Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

Пример ввода

Object\_root

Object\_root Object\_1

Object\_root Object\_2

Object\_root Object\_3

Object\_3 Object\_4

Object\_3 Object\_5

Object\_6 Object\_6

Дерево объектов, которое будет построено по данному примеру:

Object\_root

Object\_1

Object\_2

Object\_3

Object\_4

Object\_5

## 1.2 Описание выходных данных

**Первая строка:**

«имя корневого объекта»

**Вторая строка и последующие строки** имена головного и подчиненных объектов очередного уровня разделенных двумя пробелами.

«имя головного объекта» «имя подчиненного объекта»[[ «имя подчиненного объекта»] …….]

Пример вывода:

Object\_root

Object\_root Object\_1 Object\_2 Object\_3

Object\_3 Object\_4 Object\_5

# 2 МЕТОД РЕШЕНИЯ

Для решения задачи используются:

* Библиотеки iostream, string, vector
* Пространство имен std
* Оператор цикла со счётчиком for
* Условная конструкция if\_else
* Объекты стандартных потоков ввода/вывода cin/cout.

Таблица 1. Таблица иерархии классов программы.

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\_derived | public | |  | 3 |  |
| 2 | | cl\_application |  | |  | Класс корневого объекта (приложения). |  |  |
| 3 | | cl\_derived |  | |  | Класс объектов, подчинённых корневому объекту класса Application. |  |  |

* Класс cl\_base:
  + Свойства/поля:
    - Поле, отвечающее за наименование объекта:
      * Наименование - name;
      * Тип - строковый;
      * Модификатор доступа - закрытый;
    - Поле, отвечающее за хранение указателя на родительский объект:
      * Наименование - parent;
      * Тип - cl\_base\*;
      * Модификатор доступа - закрытый;
    - Поле, отвечающее за хранение подчиненных объектов:
      * Наименование - children;
      * Тип - вектор типа cl\_base\*;
      * Модификатор доступа - закрытый;
  + Методы:
    - Конструктор cl\_base
      * Функционал - Параметризированный конструктор с параметром указателя на головной объект в дереве иерархии и наименованием объекта;
    - Деструктор ~cl\_base
      * Функционал - Деструктор для удаления подчиненных объектов текущего объекта;
    - Метод SetName
      * Функционал - Используется для редактирования наименования объекта;
    - Метод GetName
      * Функционал - Используется для получения наименования объекта;
    - Метод GetParent
      * Функционал - Используется для получения указателя на головной объект текущего объекта;
    - Метод PrintObjectTree
      * Функционал - Используется для вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
    - Метод GetChild
      * Функционал - Используется для получения указателя на подчиненный объект текущего объекта по его наименованию;
* Класс cl\_application:
  + Методы:
    - Конструктор cl\_application
      * Функционал - Параметризированный конструктор с параметром указателя на головной объект в дереве иерархии;
    - Метод build\_tree\_objects
      * Функционал - Используется для построения исходного дерева иерархии объектов (конструирования моделируемой системы);
    - Метод exec\_app
      * Функционал - Используется для запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи);
* Класс cl\_derived:
  + Методы:
    - Конструктор cl\_derived
      * Функционал -Параметризированный конструктор с параметром 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 конструктора класса cl\_base

Функционал: Создание объекта на основе класса cl\_base.

Параметры: cl\_base\* parent - указатель на родительский объект создаваемого объекта, string name - наименование создаваемого объекта, с значением по умолчанию "Base object".

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Присваивание полю parent создаваемого объекта значение параметра parent | 2 |
| 2 |  | Присваивание полю name создаваемого объекта значение параметра name | 3 |
| 3 | Значение выполнения метода GetParent не nullptr | Добавление в children, список родительского объекта для подчинённых объектов, значение создаваемого объекта | ∅ |
|  |  | ∅ |

## Алгоритм деструктора класса cl\_base

Функционал: Удаление текущего объекта и его подчинённых объектов из памяти.

Параметры: Отсутствуют.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление переменной child универсального типа, отвечающей за хранение отдельного объекта списка children | 2 |
| 2 | child находится в списке children | Удаление значения child из памяти с помощью оператора delete | 3 |
|  |  | ∅ |
| 3 |  | Переход к следующему объекту списка children | 2 |

## Алгоритм метода SetName класса cl\_base

Функционал: Присваивание нового значения наименованию текущего объекта.

Параметры: string newName - новое значение наименования объекта.

Возвращаемое значение: bool - индикатор корректности изменения наименования объекта.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода SetName класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Значение выполнения метода GetParent и значение выполнения метода GetChild объекта, полученного из метода GetParent, не равно nullptr | Возврат значения false | ∅ |
|  | Присваивание полю name текущего объекта значение параметра newName | 2 |
| 2 |  | Возврат значения true | ∅ |

## Алгоритм метода GetName класса cl\_base

Функционал: Возврат значения наименования текущего объекта.

Параметры: Отсутствуют.

Возвращаемое значение: string - значение имени объекта.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода GetName класса cl\_base

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

## Алгоритм метода GetParent класса cl\_base

Функционал: Возврат указателя на родительский объект текущего объекта.

Параметры: Отсутствуют.

Возвращаемое значение: cl\_base\* - указатель на возвращаемый родительский объект текущего объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода GetParent класса cl\_base

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

## Алгоритм метода PrintObjectTree класса cl\_base

Функционал: Вывод иерархии объектов текущего объекта на экран.

Параметры: Отсутствуют.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода PrintObjectTree класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | Список children текущего объекта не пустой | Переход на новую строку и вывод поля name текущего объекта | 2 |
|  |  | ∅ |
| 2 |  | Объявление перемененной-объекта child универсального типа, отвечающей за хранение отдельного объекта списка children | 3 |
| 3 | child находится в списке children | Вывод двух пробелов и поля name переменной-объекта child на ээкран | 4 |
|  |  | 5 |
| 4 |  | Переход к следующему объекту списка children | 3 |
| 5 |  | Вызов метода PrintObjectTree() последнего элемента списка children текущего объекта | ∅ |

## Алгоритм метода GetChild класса cl\_base

Функционал: Возвращение указателся на искомый подчиненный объект по его наименованию.

Параметры: string objName - наименование искомого подчиненного объекта.

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

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода GetChild класса cl\_base

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление переменной-объекта child  универсального типа, отвечающей за хранение  отдельного объекта списка | 2 |
| 2 | child находится в списке  children |  | 3 |
|  |  | 5 |
| 3 | Значение поля name  переменной-объекта child  равно значению параметра  objName | Возврат значения переменной-объекта child | ∅ |
|  |  | 4 |
| 4 |  | Переход к следующему объекту списка children | 2 |
| 5 |  | Возврат nullptr | ∅ |

## Алгоритм конструктора класса cl\_application

Функционал: Создание головного объекта дерева иерархии объектов.

Параметры: cl\_base\* parent - указатель на родительский объект создаваемого объекта.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl\_application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с  аргументом parent, являющимся параметром конструктора  производного класса cl\_application | ∅ |

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

Функционал: Построение исходного 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 |  | Инициализация объекта parentObj класса cl\_base  типа указатель на базовый класс значением типа  указатель на текущий объект | 2 |
| 2 |  | Инициализация объекта childObj класса cl\_base  типа указатель на базовый класс значением nullptr | 3 |
| 3 |  | Объявление переменных headName и childName  типа string | 4 |
| 4 |  | Ввод значения переменной headName | 5 |
| 5 |  | Вызов метода SetName с аргументом headName  текущего объекта | 6 |
| 6 | Условие цикла истинно | Ввод значений переменных headName и childName | 7 |
|  | Выход из цикла | ∅ |
| 7 | Значение переменной  headName равно childName | Выход из цикла | ∅ |
|  |  | 8 |
| 8 | Значение объекта childObj не  равно nullptr и значение  переменной headName равно  наименованию объекта  childObj | Присваивание объекту parentObj значение объекта  childObj | 9 |
|  |  | 9 |
| 9 | Значение переменной headName равно  наименованию объекта  parentObj и объект,  находящийся в списке  подчиненных объектов  parentObj, с наименованием  childName равен nullptr | Присваивание объекту childObj значение создаваемого объекта класса cl\_1, путём вызова  параметризированного конструктора, через  оператор new | 6 |
|  |  | 6 |

## Алгоритм метода exec\_app класса cl\_application

Функционал: Запуск приложения и вывод всего дерева иерархии объектов на экран.

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности выполнения метода.

Алгоритм метода представлен в таблице 11.

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

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод результата выполнения метода GetName текущего объекта на  экран | 2 |
| 2 |  | Вызов метода PrintObjectTree текущего объекта | 3 |
| 3 |  | Возвращение значения работы метода 0 | ∅ |

## Алгоритм конструктора класса cl\_1

Функционал: Создание объекта на основе класса cl\_1.

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\_1

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с  аргументом parent, являющимся параметром конструктора  производного класса cl\_1 | ∅ |

## Алгоритм функции main

Функционал: Основной алгоритм работы программы.

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения программы.

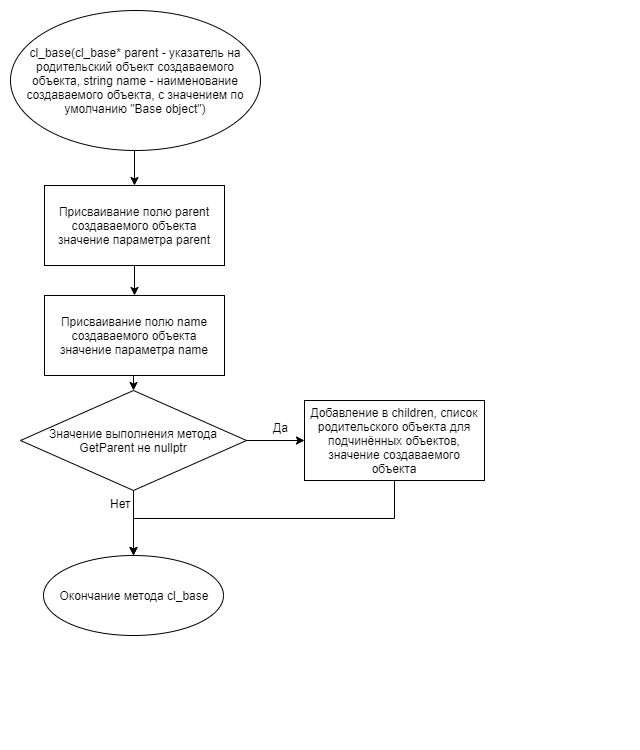
Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции main

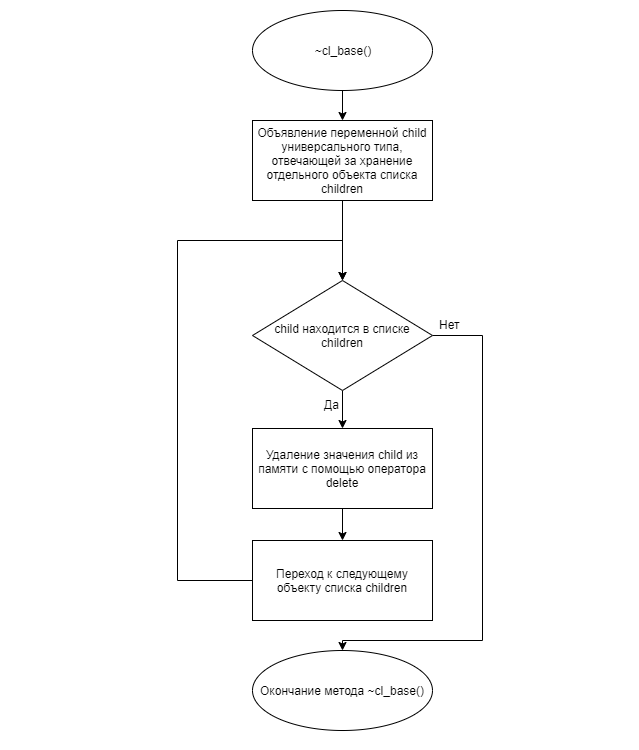
| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Создание объекта ob\_cl\_application класса cl\_application с аргументом,  равным nullptr | 2 |
| 2 |  | Вызов метода build\_tree\_objects объекта ob\_cl\_application | 3 |
| 3 |  | Вызов метода exec\_app объекта ob\_cl\_application | 4 |
| 4 |  | Возврат результата работы метода exec\_app | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

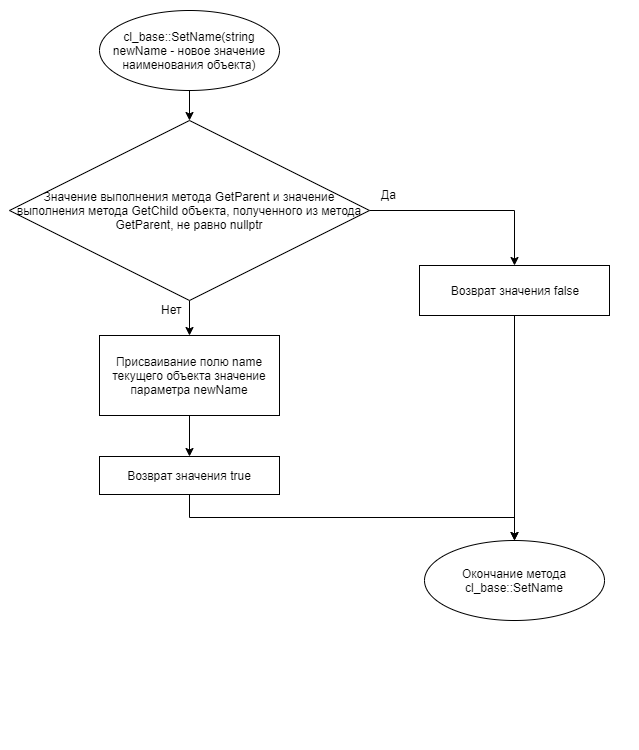
Представим описание алгоритмов в графическом виде на рисунках 1-11.



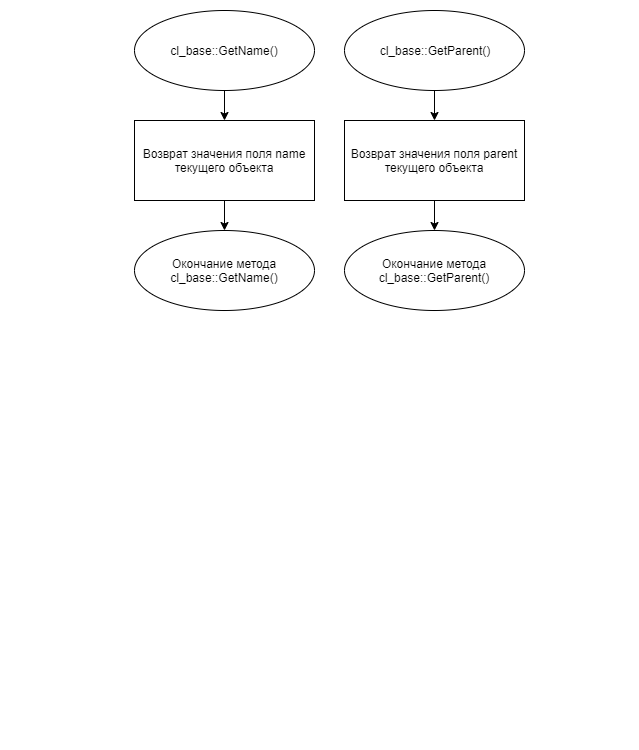
**Рисунок 1 – Блок-схема алгоритма**



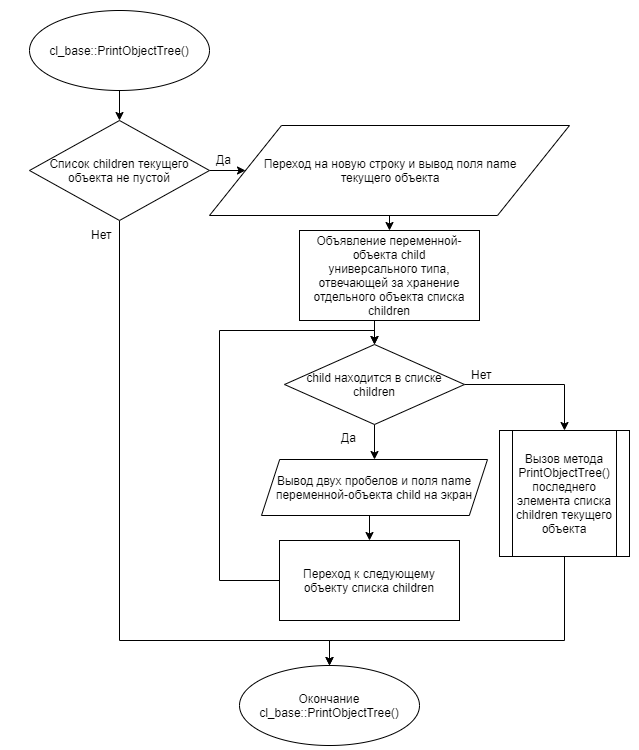
**Рисунок 2 – Блок-схема алгоритма**



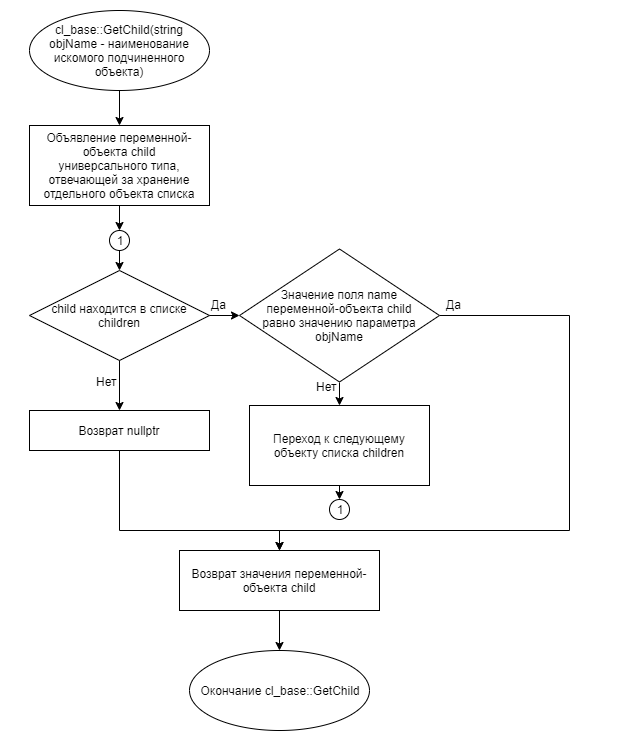
**Рисунок 3 – Блок-схема алгоритма**



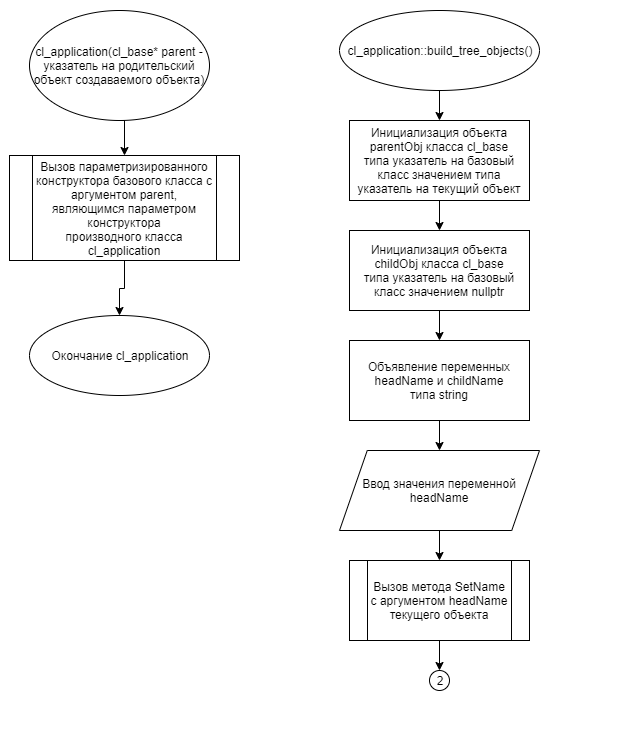
**Рисунок 4 – Блок-схема алгоритма**



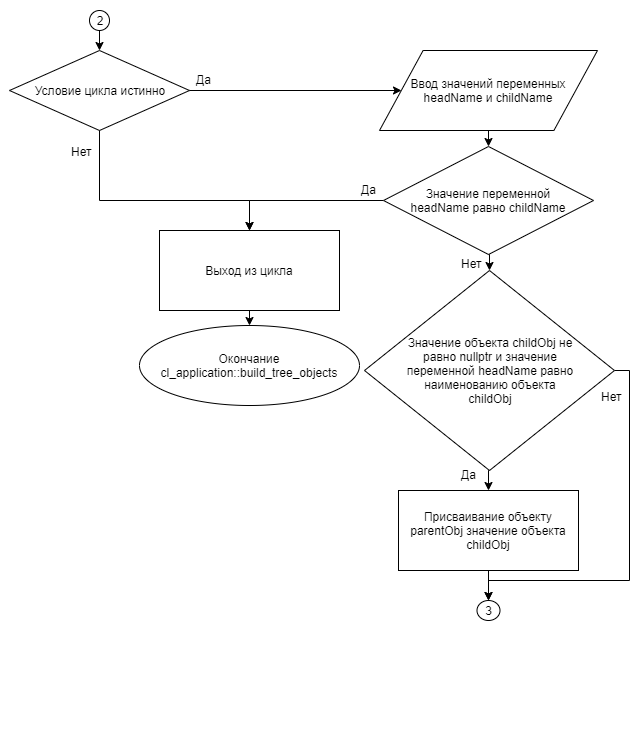
**Рисунок 5 – Блок-схема алгоритма**



**Рисунок 6 – Блок-схема алгоритма**



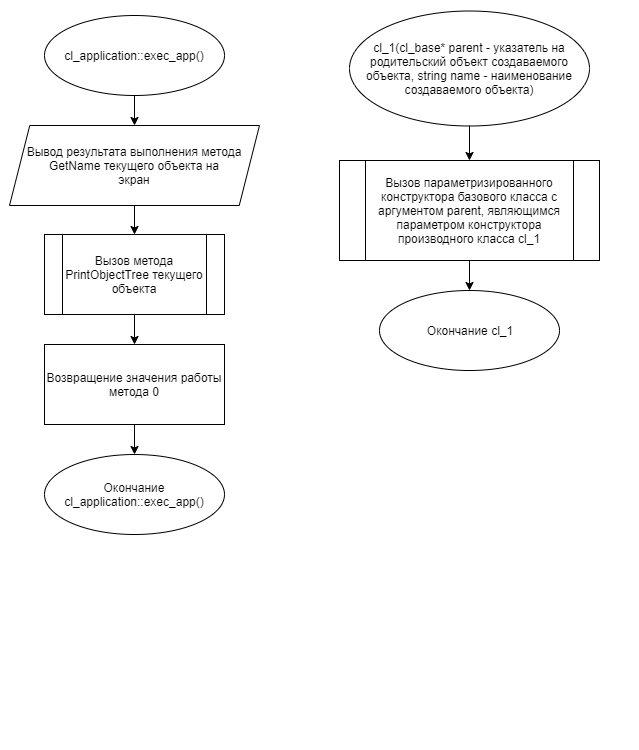
**Рисунок 7 – Блок-схема алгоритма**



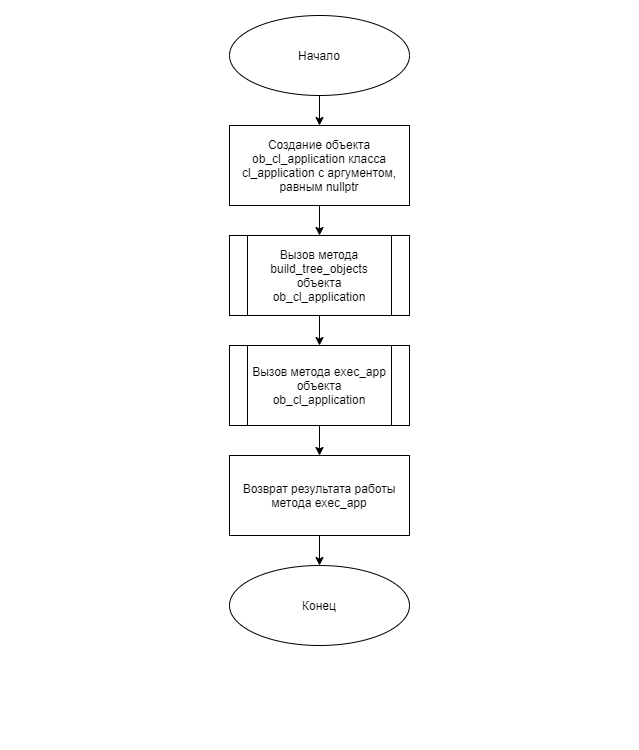
**Рисунок 8 – Блок-схема алгоритма**



**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"  #include <string>  class cl\_1 : public cl\_base  {  public:  cl\_1(cl\_base\* parent, string name);  };  #endif |

## **Файл** **cl\_application.cpp**

Листинг 3 – cl\_application.cpp

|  |
| --- |
| #include "cl\_application.h"  cl\_application::cl\_application(cl\_base\* parent) : cl\_base(parent)  {}  void cl\_application::build\_tree\_objects()  {  cl\_base\* parentObj = this;  cl\_base\* childObj = nullptr;  string headName, childName;  cin >> headName;  SetName(headName);  while(true)  {  cin >> headName >> childName;  if (headName == childName)  {  break;  }  if (childObj != nullptr && headName == childObj->GetName())  {  parentObj = childObj;  }  if (headName == parentObj->GetName() && parentObj->GetChild(childName) == nullptr)  {  childObj = new cl\_1(parentObj, childName);  }  }  }  int cl\_application::exec\_app()  {  cout << GetName();  PrintObjectTree();  return 0;  } |

## **Файл** **cl\_application.h**

Листинг 4 – cl\_application.h

|  |
| --- |
| #ifndef \_\_CL\_APPLICATION\_\_H  #define \_\_CL\_APPLICATION\_\_H  #include "cl\_base.h"  #include "cl\_1.h"  #include <iostream>  #include <string>  using namespace std;  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 5 – 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;  }  void cl\_base::PrintObjectTree() const  {  if (!children.empty())  {  cout << endl << GetName();  for (auto child : children)  {  cout << " " << child->GetName();  }  children[children.size() - 1]->PrintObjectTree();  }  }  cl\_base\* cl\_base::GetChild(string objName) const  {  for (auto child: children)  {  if (child->GetName() == objName)  {  return child;  }  }  return nullptr;  } |

## **Файл** **cl\_base.h**

Листинг 6 – cl\_base.h

|  |
| --- |
| #ifndef \_\_CL\_BASE\_\_H  #define \_\_CL\_BASE\_\_H  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class cl\_base  {  string name;  cl\_base\* parent;  vector<cl\_base\*> children;  public:  cl\_base(cl\_base\* parent, string name = "Base\_object");  ~cl\_base();  bool SetName(string newName);  string GetName() const;  cl\_base\* GetParent() const;  void PrintObjectTree() const;  cl\_base\* GetChild(string objName) const;  };  #endif |

## **Файл** **main.cpp**

Листинг 7 – 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 14.

Таблица 14 – Результат тестирования программы

| Входные данные | Ожидаемые выходные данные | Фактические выходные данные |
| --- | --- | --- |
| Object\_root  Object\_root Object\_1  Object\_root Object\_2  Object\_root Object\_3  Object\_root Object\_4  Object\_4 Object\_5  Object\_4 Object\_6  Object\_6 Object\_7  Object\_8 Object\_8 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 |
| Object\_root  Object\_root Object\_1  Object\_root Object\_2  Object\_root Object\_3  Object\_root Object\_4  Object\_3 Object\_10  Object\_4 Object\_5  Object\_4 Object\_6  Object\_6 Object\_7  Object\_8 Object\_8 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 |
| Object\_root  Object\_root Object\_1  Object\_root Object\_2  Object\_root Object\_3  Object\_root Object\_4  Object\_3 Object\_2  Object\_4 Object\_5  Object\_4 Object\_6  Object\_6 Object\_7  Object\_8 Object\_8 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 |
| Object\_root  Object\_root Object\_1  Object\_root Object\_2  Object\_root Object\_3  Object\_root Object\_1  Object\_root Object\_4  Object\_3 Object\_2  Object\_4 Object\_5  Object\_4 Object\_6  Object\_6 Object\_7  Object\_6 Object\_2  Object\_8 Object\_8 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 Object\_2 | Object\_root  Object\_root Object\_1 Object\_2 Object\_3 Object\_4  Object\_4 Object\_5 Object\_6  Object\_6 Object\_7 Object\_2 |
| Object\_root  Object\_root Object\_1  Object\_root Object\_2  Object\_2 Object\_9  Object\_root Object\_3  Object\_root Object\_4  Object\_4 Object\_5  Object\_4 Object\_6  Object\_6 Object\_7  Object\_8 Object\_8 | Object\_root  Object\_root Object\_1 Object\_2  Object\_2 Object\_9 | Object\_root  Object\_root Object\_1 Object\_2  Object\_2 Object\_9 |
| ob\_1  ob\_1 ob\_2  ob\_1 ob\_3  ob\_3 ob\_5  ob\_3 ob\_6  ob\_6 ob\_7  ob\_6 ob\_8  ob\_6 ob\_6 | ob\_1  ob\_1 ob\_2 ob\_3  ob\_3 ob\_5 ob\_6  ob\_6 ob\_7 ob\_8 | ob\_1  ob\_1 ob\_2 ob\_3  ob\_3 ob\_5 ob\_6  ob\_6 ob\_7 ob\_8 |
| ob\_1  ob\_1 ob\_2  ob\_1 ob\_3  ob\_3 ob\_5  ob\_3 ob\_6  ob\_5 ob\_7  ob\_5 ob\_8  ob\_6 ob\_6 | ob\_1  ob\_1 ob\_2 ob\_3  ob\_3 ob\_5 ob\_6 | ob\_1  ob\_1 ob\_2 ob\_3  ob\_3 ob\_5 ob\_6 |
| ob\_1  ob\_1 ob\_2  ob\_1 ob\_3  ob\_3 ob\_5  ob\_3 ob\_6  ob\_6 ob\_6 | ob\_1  ob\_1 ob\_2 ob\_3  ob\_3 ob\_5 ob\_6 | ob\_1  ob\_1 ob\_2 ob\_3  ob\_3 ob\_5 ob\_6 |
| ob\_1  ob\_1 ob\_2  ob\_1 ob\_3  ob\_2 ob\_5  ob\_2 ob\_6  ob\_6 ob\_6 | ob\_1  ob\_1 ob\_2 ob\_3 | ob\_1  ob\_1 ob\_2 ob\_3 |
| ob\_1  ob\_1 ob\_2  ob\_1 ob\_2  ob\_1 ob\_3  ob\_1 ob\_1 | ob\_1  ob\_1 ob\_2 ob\_3 | ob\_1  ob\_1 ob\_2 ob\_3 |
| ob\_1  ob\_1 ob\_2  ob\_1 ob\_3  ob\_2 ob\_5  ob\_2 ob\_6  ob\_1 ob\_ 6 | ob\_1  ob\_1 ob\_2 ob\_3 |  |

# 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).