Здесь будет титульник, листай ниже

**СОДЕРЖАНИЕ**

[1 ПОСТАНОВКА ЗАДАЧИ 6](#__RefHeading___Toc5238_907792531)

[1.1 Описание входных данных 8](#__RefHeading___Toc5240_907792531)

[1.2 Описание выходных данных 8](#__RefHeading___Toc5242_907792531)

[2 МЕТОД РЕШЕНИЯ 10](#__RefHeading___Toc5244_907792531)

[3 ОПИСАНИЕ АЛГОРИТМОВ 13](#__RefHeading___Toc5246_907792531)

[3.1 Алгоритм конструктора класса ClassBase 13](#__RefHeading___Toc5248_907792531)

[3.2 Алгоритм деструктора класса ClassBase 13](#__RefHeading___Toc5250_907792531)

[3.3 Алгоритм метода setClassName класса ClassBase 14](#__RefHeading___Toc5252_907792531)

[3.4 Алгоритм метода getClassName класса ClassBase 15](#__RefHeading___Toc5254_907792531)

[3.5 Алгоритм метода getParent класса ClassBase 15](#__RefHeading___Toc5256_907792531)

[3.6 Алгоритм метода outputTree класса ClassBase 15](#__RefHeading___Toc5258_907792531)

[3.7 Алгоритм метода getChild класса ClassBase 16](#__RefHeading___Toc5260_907792531)

[3.8 Алгоритм конструктора класса ClassObj 17](#__RefHeading___Toc5262_907792531)

[3.9 Алгоритм метода buildTreeObjects класса ClassApp 17](#__RefHeading___Toc5264_907792531)

[3.10 Алгоритм метода execApp класса ClassApp 19](#__RefHeading___Toc5266_907792531)

[3.11 Алгоритм конструктора класса ClassApp 19](#__RefHeading___Toc5268_907792531)

[3.12 Алгоритм функции main 20](#__RefHeading___Toc5270_907792531)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 21](#__RefHeading___Toc5272_907792531)

[5 КОД ПРОГРАММЫ 28](#__RefHeading___Toc5274_907792531)

[5.1 Файл ClassApp.cpp 28](#__RefHeading___Toc5276_907792531)

[5.2 Файл ClassApp.h 29](#__RefHeading___Toc5278_907792531)

[5.3 Файл ClassBase.cpp 29](#__RefHeading___Toc5280_907792531)

[5.4 Файл ClassBase.h 30](#__RefHeading___Toc5282_907792531)

[5.5 Файл ClassObj.cpp 31](#__RefHeading___Toc5284_907792531)

[5.6 Файл ClassObj.h 31](#__RefHeading___Toc5286_907792531)

[5.7 Файл main.cpp 31](#__RefHeading___Toc5288_907792531)

[6 ТЕСТИРОВАНИЕ 33](#__RefHeading___Toc5290_907792531)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 35](#__RefHeading___Toc5292_907792531)

# 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

Программа использует стандартное пространство имён, объекты string, vector, стандартные потоки ввода/вывода cin/cout.

* Таблица 1 – Иерархия наследования классов

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | | Имя класса | Классы-наследники | | Модификатор доступа при наследовании | Описание | Номер | Комментарий |
| 1 | | ClassBase |  | |  | Базовый класс в иерархии классов. Содержит основные поля и методы. |  |  |
|  | | ClassApp | public | |  | 2 |  |
|  | | ClassObj | public | |  | 3 |  |
| 2 | | ClassApp |  | |  | Класс корневого объекта (приложения). |  |  |
| 3 | | ClassObj |  | |  | Класс объектов, подчинённых корневому объекту класса ClassApp. |  |  |

Класс ClassBase

Свойства/поля:

* Поле, отвечающее за наименование объекта
  + Наименование - className
  + Тип - строковый
  + Модификатор доступа - закрытый
* Поле, отвечающее за наименование объекта
  + Наименование - parent
  + Тип - ClassBase\*
  + Модификатор доступа - закрытый
* Поле, отвечающее за наименование объекта
  + Наименование - childs
  + Тип - вектор типа ClassBase\*
  + Модификатор доступа - закрытый

Методы:

* Конструтор ClassBase - параметризированный конструктор с параметром указателя на головной объект и наименованием объекта
* Деструктор ~ClassBase - деструктор для удаления подчинённых объектов текущего объекта
* Метод setClassName - редактирование наименования объекта
* Метод getClassName - получение значения className
* Метод getParent - получение значения parent
* Метод outputTree - вывод наименований объектов в дереве иерархии слева направо, сверху вниз
* Метод getChild - получение указателя на подчиненный объект текущего объекта по его наименованию

Класс ClassApp

Методы:

* Конструктор ClassApp - параметризированный конструктор с параметром указателя на головной объект в дереве иерархии объектов
* Метод buildTreeObject - построение исходного дерева иерархии объектов
* Метод execApp - используется для запуска приложения (начало выполнения решения задачи)

Класс ClassObj

Методы:

* Конструктор ClassObj - 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 ClassBase

Функционал: Создание объекта на основе класса ClassBase.

Параметры: ClassBase\* parent - указатель на родительский объект, string className - имя класса.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса ClassBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Определение поля parent параметром parent | 2 |
| 2 |  | Определение поля className параметром className | 3 |
| 3 | parent != nullptr | Добавление в список childs родительского объекта для подчинённых объектов значение создаваемого объекта | ∅ |
|  |  | ∅ |

## Алгоритм деструктора класса ClassBase

Функционал: Высвобождение памяти, выделенной под поле.

Параметры: Отсутствуют.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса ClassBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление переменной объекта child универсального типа, отвечающий за хранение отдельного значения объекта списка | 2 |
| 2 | child находится в списке childs | Высвобождение памяти, выделенной под объект child | 2 |
|  |  | ∅ |

## Алгоритм метода setClassName класса ClassBase

Функционал: Присваивание нового значения полю className.

Параметры: string className.

Возвращаемое значение: bool - индикатор корректности изменения значения className.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода setClassName класса ClassBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов метода getParent | 2 |
| 2 |  | Вызов метода getChild | 3 |
| 3 | Возвращаемые значения getParent и getChild не равны nullptr | Возврат значения false | ∅ |
|  | Присваивание полю className значение параметра className | 4 |
| 4 |  | Возврат значения true | ∅ |

## Алгоритм метода getClassName класса ClassBase

Функционал: Получение значения className.

Параметры: Отсутствуют.

Возвращаемое значение: string - поле className.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода getClassName класса ClassBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Возвращение className | ∅ |

## Алгоритм метода getParent класса ClassBase

Функционал: Получение значения parent.

Параметры: Отсутствуют.

Возвращаемое значение: ClassBase\* - поле parent.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода getParent класса ClassBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Возвращение parent | ∅ |

## Алгоритм метода outputTree класса ClassBase

Функционал: Вывод дерева объектов.

Параметры: Отсутствуют.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода outputTree класса ClassBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | childs пустой |  | ∅ |
|  | Вывод переноса каретки и className | 2 |
| 2 |  | Объявление переменной объекта child универсального типа, отвечающий за хранение отдельного значения объекта списка | 3 |
| 3 | child находится в списке childs | Вывод двух пробелов и поля className объекта child | 3 |
|  |  | 4 |
| 4 |  | Вызов метода outputTree для последнего элемента массива childs | ∅ |

## Алгоритм метода getChild класса ClassBase

Функционал: Поиск объекта с полем className равным childName.

Параметры: string childName - название необходимого объекта.

Возвращаемое значение: ClassBase\* - объект с полем className равным childName.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода getChild класса ClassBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление переменной объекта child универсального типа, отвечающий за хранение отдельного значения объекта списка | 2 |
| 2 | child находится в списке childs |  | 3 |
|  |  | 6 |
| 3 |  | Вызов метода childName | 4 |
| 4 |  | Вызов метода getClassName | 5 |
| 5 | Результат метода childName равно результату метода getClassName | Возвращение child | ∅ |
|  |  | 2 |
| 6 |  | Возвращение nullptr | ∅ |

## Алгоритм конструктора класса ClassObj

Функционал: Создание объекта на основе класса ClassObj.

Параметры: ClassBase\* parent - указатель на родительский объект, string className - имя класса.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса ClassObj

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с аргументами parent и className, являющимися параметром  конструктора произоводного класса ClassObj | ∅ |

## Алгоритм метода buildTreeObjects класса ClassApp

Функционал: Построение исходного дерева иерархии объектов.

Параметры: Отсутствуют.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода buildTreeObjects класса ClassApp

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Объявление переменных childName и parentName типа string | 2 |
| 2 |  | Инициализация объекта PARENT класса ClassBase типа указатель на базовый класс значением типа указатель на текущий объект | 3 |
| 3 |  | Инициализация объекта CHILD класса ClassBase типа указатель на базовый класс значением nullptr | 4 |
| 4 |  | Ввод значения переменной parentName | 5 |
| 5 |  | Вызов метода setClassName с параметром parentName | 6 |
| 6 |  | Ввод значений переменных parentName и childName | 7 |
| 7 | Значение переменной parentName равно childName |  | ∅ |
|  |  | 8 |
| 8 | Значение объекта CHILD не равно nullptr и значение переменной parentName равно наименованию объекта CHILD | Присвоение объекту PARENT значение объекта CHILD | 9 |
|  |  | 9 |
| 9 | Значение переменной parentName равно наименованию объекта PARENT и объекта, находящийся в списке подчиненных объектов  PARENT, с наименованием childName равен nullptr | Присваивание объекту CHILD значение создаваемого объекта ClassObj, путём вызова параметризированного конструктора через оператор new | 6 |
|  |  | 6 |

## Алгоритм метода execApp класса ClassApp

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

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности выполнения метода.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода execApp класса ClassApp

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод результата выполнения метода getClassName текущего объекта на экран | 2 |
| 2 |  | Вызов метода outputTree | 3 |
| 3 |  | Возвращение нуля | ∅ |

## Алгоритм конструктора класса ClassApp

Функционал: Создание объекта на основе класса ClassApp.

Параметры: ClassBase\* parent - указатель на родительский объект.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса ClassApp

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вызов параметризированного конструктора базового класса с аргументом parent, являющимся параметром конструктора  произоводного класса ClassObj | ∅ |

## Алгоритм функции main

Функционал: Основной алгоритм работы программы.

Параметры: Отсутствуют.

Возвращаемое значение: int - индикатор корректности завершения программы.

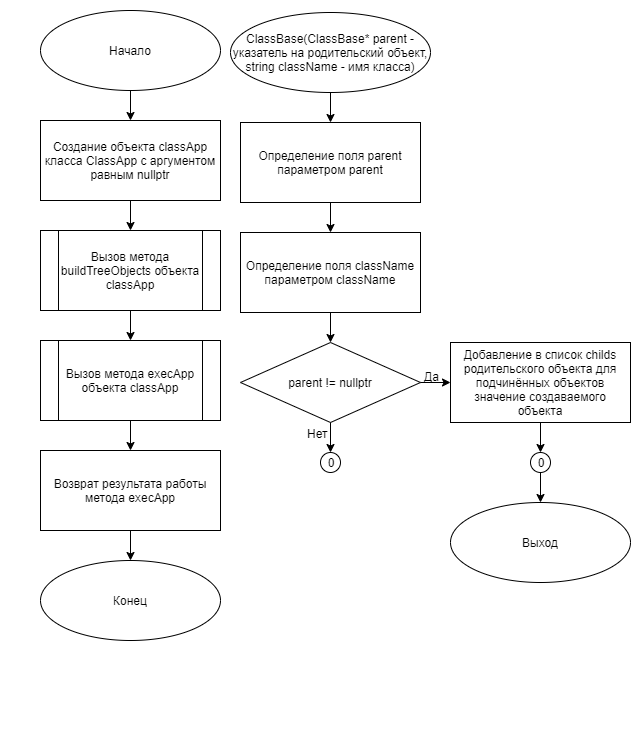
Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции main

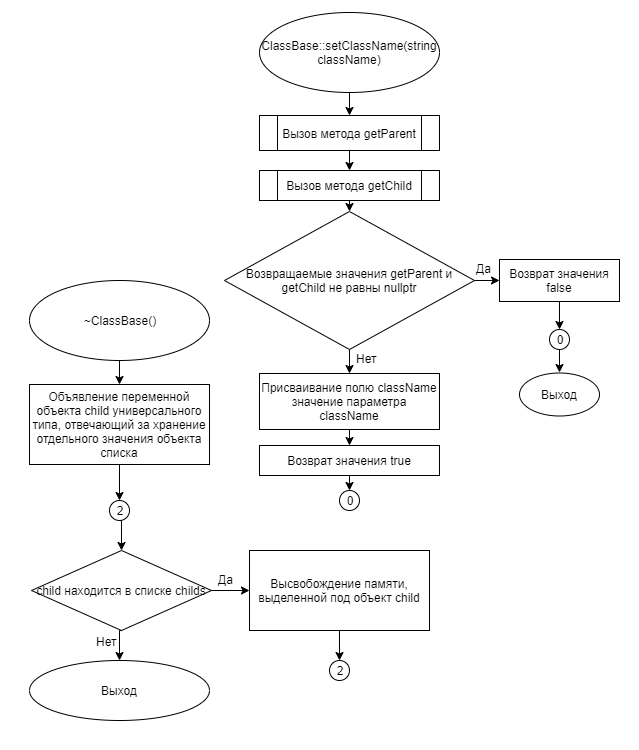
| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Создание объекта classApp класса ClassApp с аргументом равным nullptr | 2 |
| 2 |  | Вызов метода buildTreeObjects объекта classApp | 3 |
| 3 |  | Вызов метода execApp объекта classApp | 4 |
| 4 |  | Возврат результата работы метода execApp | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

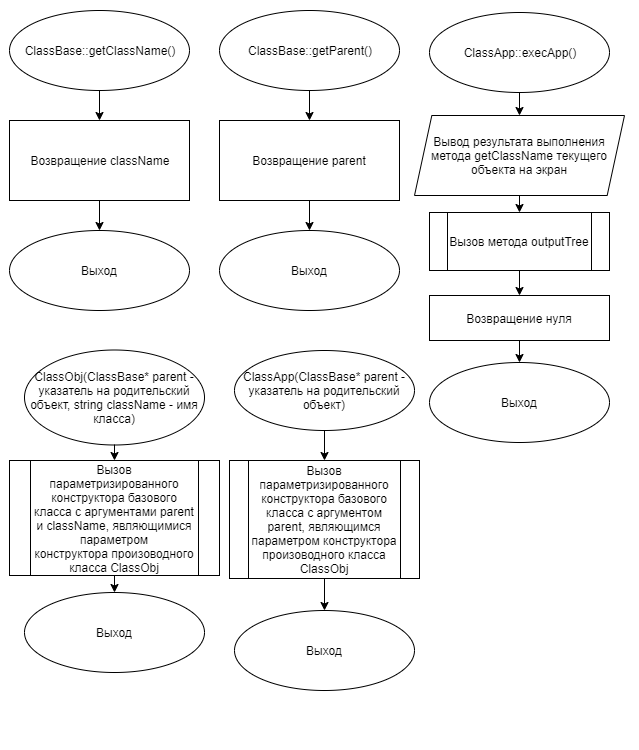
Представим описание алгоритмов в графическом виде на рисунках 1-7.



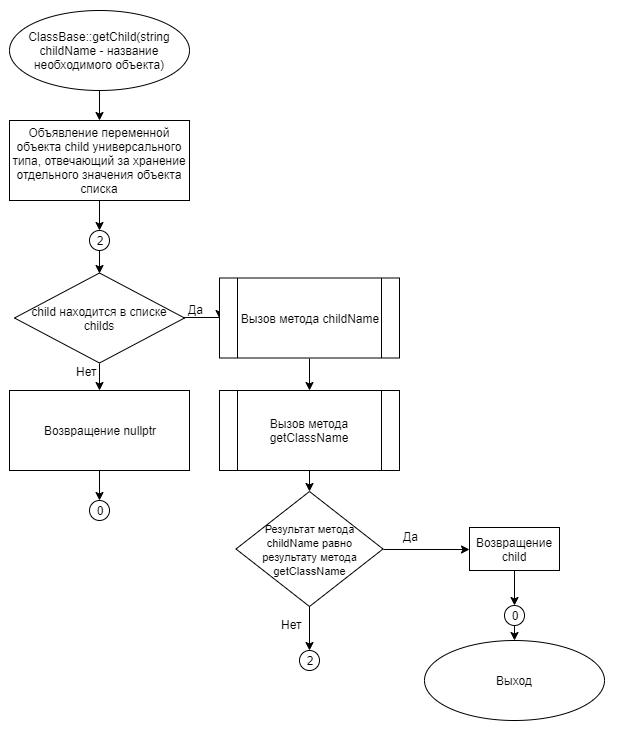
**Рисунок 1 – Блок-схема алгоритма**



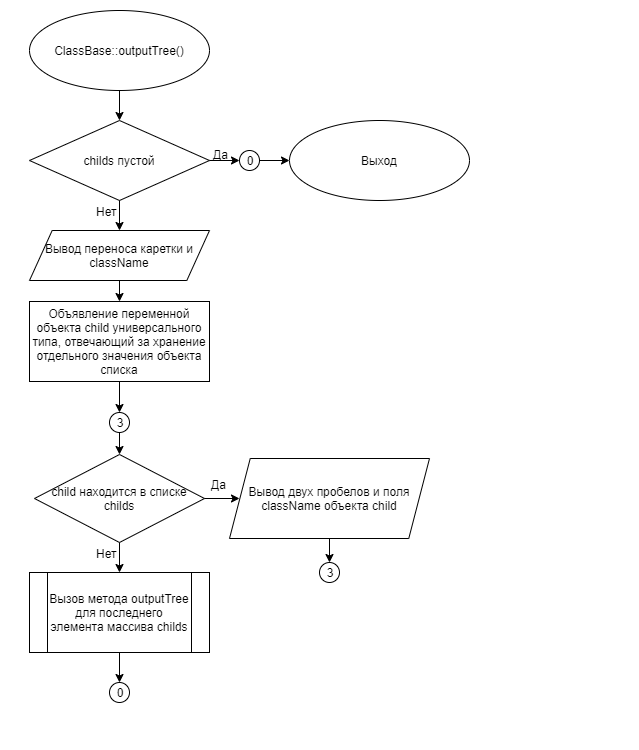
**Рисунок 2 – Блок-схема алгоритма**



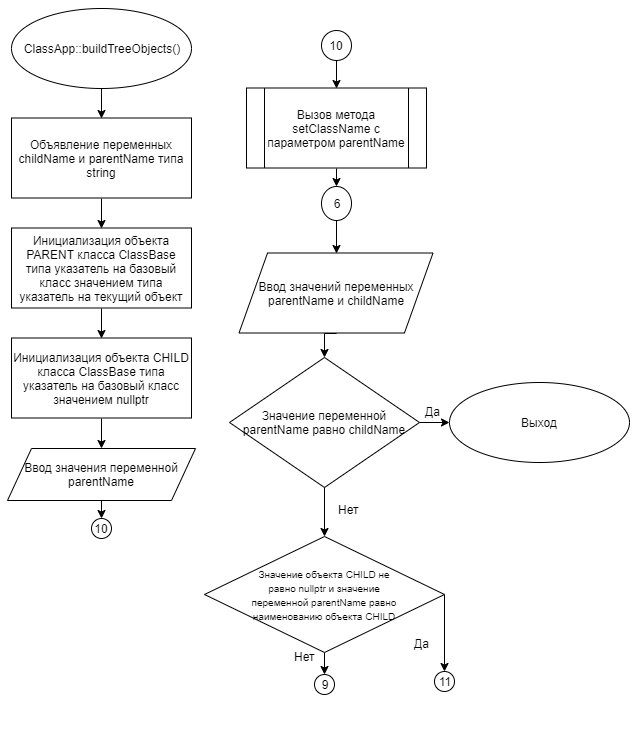
**Рисунок 3 – Блок-схема алгоритма**



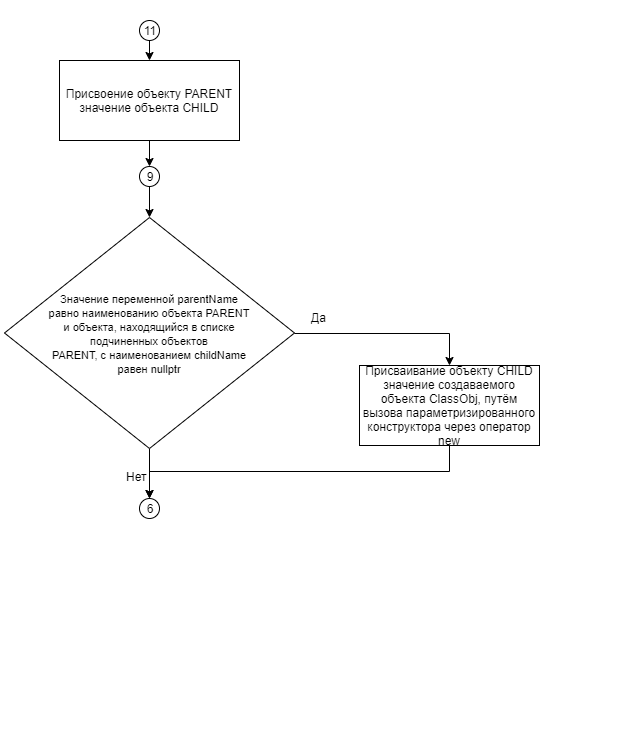
**Рисунок 4 – Блок-схема алгоритма**



**Рисунок 5 – Блок-схема алгоритма**



**Рисунок 6 – Блок-схема алгоритма**



**Рисунок 7 – Блок-схема алгоритма**

# 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

## **Файл** **ClassApp.cpp**

Листинг 1 – ClassApp.cpp

|  |
| --- |
| #include "ClassApp.h"  ClassApp::ClassApp(ClassBase\* parent):  ClassBase(parent){}  void ClassApp::buildTreeObjects()  {  string childName, parentName;  ClassBase\* PARENT = this;  ClassBase\* CHILD = nullptr;  cin >> parentName;  this -> setClassName(parentName);  while (true)  {  cin >> parentName >> childName;  if (parentName == childName)  {  break;  }  if (CHILD != nullptr && parentName == CHILD->getClassName())  {  PARENT = CHILD;  }  if (PARENT->getChild(childName) == nullptr && parentName == PARENT->getClassName())  {  CHILD = new ClassObj(PARENT, childName);  }  }  }  int ClassApp::execApp()  {  cout << getClassName();  outputTree();  return 0;  } |

## **Файл** **ClassApp.h**

Листинг 2 – ClassApp.h

|  |
| --- |
| #ifndef \_\_CLASSAPP\_\_H  #define \_\_CLASSAPP\_\_H  #include "ClassBase.h"  #include "ClassObj.h"  class ClassApp : public ClassBase  {  public:  ClassApp(ClassBase\* parent);//Конструктор, создающий объект класса ClassApp  void buildTreeObjects();//Метод, создающий иерархию объектов  int execApp();//Метод, запускающий алгоритм программы  };  #endif |

## **Файл** **ClassBase.cpp**

Листинг 3 – ClassBase.cpp

|  |
| --- |
| #include "ClassBase.h"  ClassBase::ClassBase(ClassBase\* parent, string className)  {  this->parent = parent;  this->className = className;  if (parent != nullptr)  {  parent->childs.push\_back(this);  }  }  ClassBase::~ClassBase()  {  for (auto child : childs)  {  delete child;  }  }  bool ClassBase::setClassName(string className)  {  if (getParent() != nullptr && getChild(className) !=nullptr)  {  return false;  }  this->className = className;  return true;  }  string ClassBase::getClassName()  {  return className;  }  ClassBase\* ClassBase::getParent()  {  return parent;  }  void ClassBase::outputTree()  {  if (!childs.empty())  {  cout << endl << className;  for (auto child : childs)  {  cout << " " << child->className;  }  childs[childs.size()-1]->outputTree();  }  }  ClassBase\* ClassBase::getChild(string childName)  {  for (auto child : childs)  {  if (child->getClassName() == childName)  {  return child;  }  }  return nullptr;  } |

## **Файл** **ClassBase.h**

Листинг 4 – ClassBase.h

|  |
| --- |
| #ifndef \_\_CLASSBASE\_\_H  #define \_\_CLASSBASE\_\_H  #include <iostream>  #include <string>  #include <vector>  using namespace std;  class ClassBase  {  public:  ClassBase(ClassBase\* parent, string className =  "Base\_object");//Конструктор, создающий объект класса ClassBase  ~ClassBase();//Деструктор, уничтожающий объект  bool setClassName(string className);//Установка имени объекта  string getClassName();//Получение имени объекта  ClassBase\* getParent();//Получение родителя объекта  void outputTree();//Вывод дерева иерархии  ClassBase\* getChild(string childName);//Получение ребенка объекта  private:  string className;//Имя объекта  ClassBase\* parent = nullptr;//Указатель на родителя  vector<ClassBase\*> childs;//Вектор детей объекта  };  #endif |

## **Файл** **ClassObj.cpp**

Листинг 5 – ClassObj.cpp

|  |
| --- |
| #include "ClassObj.h"  ClassObj::ClassObj(ClassBase\* parent, string ClassName):  ClassBase(parent, ClassName)  {} |

## **Файл** **ClassObj.h**

Листинг 6 – ClassObj.h

|  |
| --- |
| #ifndef \_\_CLASSOBJ\_\_H  #define \_\_CLASSOBJ\_\_H  #include "ClassBase.h"  class ClassObj : public ClassBase  {  public:  ClassObj(ClassBase\* parent, string ClassName);//Конструктор, создающий объект класса ClassObj  };  #endif |

## **Файл** **main.cpp**

Листинг 7 – main.cpp

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include "ClassApp.h"  int main()  {  ClassApp classApp(nullptr);  classApp.buildTreeObjects();  return classApp.execApp();  } |

# 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\_1 | ob\_1  ob\_1 ob\_2 ob\_3 | ob\_1  ob\_1 ob\_2 ob\_3 |

# 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).