**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#__RefHeading___Toc7125_1474416929)

[1 ПОСТАНОВКА ЗАДАЧИ 7](#__RefHeading___Toc7127_1474416929)

[1.1 Описание входных данных 8](#__RefHeading___Toc7129_1474416929)

[1.2 Описание выходных данных 10](#__RefHeading___Toc7131_1474416929)

[2 МЕТОД РЕШЕНИЯ 12](#__RefHeading___Toc7133_1474416929)

[3 ОПИСАНИЕ АЛГОРИТМОВ 14](#__RefHeading___Toc7135_1474416929)

[3.1 Алгоритм метода findByPath класса CBase 14](#__RefHeading___Toc7137_1474416929)

[3.2 Алгоритм метода childExists класса CBase 16](#__RefHeading___Toc7139_1474416929)

[3.3 Алгоритм метода buildTree класса Application 17](#__RefHeading___Toc7141_1474416929)

[3.4 Алгоритм метода execute класса Application 19](#__RefHeading___Toc7143_1474416929)

[3.5 Алгоритм функции main 20](#__RefHeading___Toc7145_1474416929)

[4 БЛОК-СХЕМЫ АЛГОРИТМОВ 21](#__RefHeading___Toc7147_1474416929)

[5 КОД ПРОГРАММЫ 33](#__RefHeading___Toc7149_1474416929)

[5.1 Файл application.cpp 33](#__RefHeading___Toc7151_1474416929)

[5.2 Файл application.h 34](#__RefHeading___Toc7153_1474416929)

[5.3 Файл cbase.cpp 35](#__RefHeading___Toc7155_1474416929)

[5.4 Файл cbase.h 38](#__RefHeading___Toc7157_1474416929)

[5.5 Файл cobject.cpp 39](#__RefHeading___Toc7159_1474416929)

[5.6 Файл cobject.h 39](#__RefHeading___Toc7161_1474416929)

[5.7 Файл five.cpp 39](#__RefHeading___Toc7163_1474416929)

[5.8 Файл five.h 39](#__RefHeading___Toc7165_1474416929)

[5.9 Файл four.cpp 40](#__RefHeading___Toc7167_1474416929)

[5.10 Файл four.h 40](#__RefHeading___Toc7169_1474416929)

[5.11 Файл main.cpp 40](#__RefHeading___Toc7171_1474416929)

[5.12 Файл six.cpp 41](#__RefHeading___Toc7173_1474416929)

[5.13 Файл six.h 41](#__RefHeading___Toc7175_1474416929)

[5.14 Файл three.cpp 41](#__RefHeading___Toc7177_1474416929)

[5.15 Файл three.h 41](#__RefHeading___Toc7179_1474416929)

[5.16 Файл two.cpp 42](#__RefHeading___Toc7181_1474416929)

[5.17 Файл two.h 42](#__RefHeading___Toc7183_1474416929)

[6 ТЕСТИРОВАНИЕ 43](#__RefHeading___Toc7185_1474416929)

[ЗАКЛЮЧЕНИЕ 44](#__RefHeading___Toc7187_1474416929)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 45](#__RefHeading___Toc7189_1474416929)

# ВВЕДЕНИЕ

При моделировании работы сложных устройств и механизмов, которые включают в себя множество взаимодействующих друг с другом объектов, используют метод объектно-ориентированного программирования. Принцип которого строится [1] на том, что мы находим сходие свойства у объектов и на их основании объединяем объекты в классы.

Для того чтобы, сократить код, сделать его более универсальным и понятным используют [2] принцип наследования. Который заключается в том, что если некоторый объект совержит свойства, которые описаны в базовом классе, то для него создается новый класс, наследующий свойства класса-родителя, но также содержащий и новые свойства [3]. Объекты, постороенные по данному принципу, взаимодействуют друг с другом по прототипу "родитель - потомок". Таким образом выстраивается [4] иерархическое дерево объектов и налаживается система координирования и взаимодействия между ними.

В данной курсовой работе перед нами ставится задача разработать систему [5], которая будет действовать на описанных выше принципах. Система представляет собой дерево, состоящее из объектов, напоминающее файловую систему.

# 1 ПОСТАНОВКА ЗАДАЧИ

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

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

/                  - корневой объект;

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

.                  - текущий объект;

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

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

Примеры координат:

/

//ob\_3

.

ob\_2/ob\_3

ob\_2

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

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

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

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

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

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

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

Система отрабатывает следующие команды:

SET «координата»  – устанавливает текущий объект;

FIND «координата» – находит объект относительно текущего;

END               – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим.

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

## 1.1 Описание входных данных

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

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

После ввода состава дерева иерархии построчно вводятся команды:

SET «координата»  - установить текущий объект;

FIND «координата» - найти объект относительно текущего;

END               – завершить функционирование системы (выполнение программы).

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

Пример ввода иерархии дерева объектов.

root

/ object\_1 3

/ object\_2 2

/object\_2 object\_4 3

/object\_2 object\_5 4

/ object\_3 3

/object\_2 object\_3 6

/object\_1 object\_7 5

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

endtree

FIND object\_2/object\_4

SET /object\_2

FIND //object\_5

FIND /object\_15

FIND .

FIND object\_4/object\_7

END

## 1.2 Описание выходных данных

Первая строка:

Object tree

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

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

The head object «координата головного объекта» is not found

и прекратить работу программы.

Если дерево построено, то далее построчно:

для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

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

для команд FIND вывести:

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

Если объект не найден, то:

«искомая координата объекта»     Object is not found

Пример вывода иерархии дерева объектов.

Object tree

root

    object\_1

        object\_7

    object\_2

        object\_4

            object\_7

        object\_5

        object\_3

    object\_3

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

Object is set: object\_2

//object\_5     Object name: object\_5

/object\_15     Object is not found

.     Object name: object\_2

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

# 2 МЕТОД РЕШЕНИЯ

При решении данной задачи используется:

1. Объекты стандартного потока вывода и ввода;
2. Условный оператор if/else;
3. Оператор цикла со счетчиком for;
4. Оператор цикла с предусловием while;
5. Oбъект root классa Application
6. Объекты классов Two, Three, Four, Five, Six
7. Класс CBase:
   * Методы:
     + Метод получения указателя по пути
       1. Наименование - findByPath
       2. Параметр - строковая переменная path-путь
       3. Функционал - поиск объекта по пути от определенного объекта
       4. Тип возвращаемых данных - указатель на объект класса CBase, или nullptr, если объекта не существует
       5. Модификатор доступа - public
     + Метод проверки на существование подчиненного объекта у текущего
       1. Наименование - childExists
       2. Параметр - строковая переменная name - имя объекта для проверки
       3. Функционал - проверка всех подчиненных объектов текущего на совпадение имени поля с именем из параметра
       4. Тип возвращаемых данных - указатель на объект класса CBase, или nullptr, если объекта не существует
       5. Модификатор доступа - public
8. Класс Application:

Методы

Метод построения дерева иерархии

Наименование - buildTree

Парамерты - отсутствуют

Функционал - создание объектов и размещение в дереве

Тип возвращаемых данных - ничего не возвращает

Модификатор доступа - public

Метод запуска системы

Наименование - execute

Парамерты - отсутствуют

Функционал - создание объектов и размещение в дереве

Тип возвращаемых данных - 0 - показатель успешного выполнения программы

Модификатор доступа - public

# 3 ОПИСАНИЕ АЛГОРИТМОВ

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

## Алгоритм метода findByPath класса CBase

Функционал: Поиск объекта по пути.

Параметры: std::string path - путь к объекту.

Возвращаемое значение: CBase\* указатель на найденный объект или nullptr.

Алгоритм метода представлен в таблице 1.

Таблица 1 – Алгоритм метода findByPath класса CBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 | значение переменной path == пустой строке | вернуть nullptr | ∅ |
|  |  | 2 |
| 2 |  | инициализация целочисленной переменной state = 0 - состояние чтение пути, будет принимать значение из множества {0, 1, 2, 3, 4, 5, 6} | 3 |
| 3 |  | инициализация целочисленной переменной счетчика i = 0 | 4 |
| 4 | счетчик i < количества символов в строке path | инициализация символьной перепенной ch = path[i] | 5 |
|  |  | 18 |
| 5 | значение state == 0 |  | 6 |
|  |  | 8 |
| 6 | значение ch == '/' | присвоение state = 1 - переходим в состояние 1 на следующей итерации цикла | 17 |
|  |  | 7 |
| 7 | значение ch == '.' | присвоение state = 2 - переходим в состояние 2 на следующей итерации цикла | 17 |
|  | присвоение state = 3 - переходим в состояние 3 на следующей итерации цикла | 17 |
| 8 | значение state == 1 |  | 9 |
|  |  | 10 |
| 9 | значение ch == '/' | присвоение state = 4 - переходим в состояние 4 на следующей итерации цикла | 17 |
|  | присвоение state = 5 - переходим в состояние 5 на следующей итерации цикла | 17 |
| 10 | значение state == 2 |  | ∅ |
|  |  | 11 |
| 11 | значение state == 3 |  | 12 |
|  |  | 13 |
| 12 | значение ch == '/' | присвоение state = 6 - переходим в состояние 6 на следующей итерации цикла | 17 |
|  |  | 17 |
| 13 | значение state == 4 | вернуть значение полученное вызовом функции findByName() с параметром подстрокой path без двух первых сиволов '//' | ∅ |
|  |  | 14 |
| 14 | значение state == 5 | верунть значение полученное рекурсивным вызовом функции от корневого объекта, полученного вызовом функции getRoot(), с параметром подстрокой path без первого сивола '/' | ∅ |
|  |  | 15 |
| 15 | значение state == 6 | инициализация указателя на CBase tmp = значению полученным вызовом функции childExists() c параметром подстрокой path, обрезанной после (i-1) символа | 16 |
|  |  | 17 |
| 16 | значение указателя tmp != nullptr | вернуть значение полученное рекурсивным вызовом функции с параметром подстрокой path, обрезанной до i -го символа | ∅ |
|  | вернуть nullptr | ∅ |
| 17 |  | инкремент переменной-счетчика i | 4 |
| 18 | значение state == 1 | вернуть значение полученное вызовом функции getRoot() - корневой объект | ∅ |
|  |  | 19 |
| 19 | значение state == 2 | вернуть указатель на текущий объект | ∅ |
|  |  | 20 |
| 20 | значение state == 3 | вернуть значение полученное вызовом функции childExist() c параметром path | ∅ |
|  | вернуть nullptr | ∅ |

## Алгоритм метода childExists класса CBase

Функционал: Проверка на существование подчиненного объекта с именем name у текущего.

Параметры: string name - имя объекта.

Возвращаемое значение: CBase\* - указатель на найденный объект или nullptr -.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода childExists класса CBase

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | инициализация целочисленной переменной счетчика i = 0 | 2 |
| 2 | значение счетчика i < размера вектрора children |  | 3 |
|  |  | 5 |
| 3 | значение поля name у объекта children[i] == значению параметра name | вернуть значение children[i] | ∅ |
|  |  | 4 |
| 4 |  | инкремент переменной i | 2 |
| 5 |  | вернуть nullptr | ∅ |

## Алгоритм метода buildTree класса Application

Функционал: Построение дерева объектов.

Параметры: -.

Возвращаемое значение: -.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода buildTree класса Application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | инициализация скроковых переменных parentPath и childName | 2 |
| 2 |  | инициализация указателя на CBase tmp | 3 |
| 3 |  | инициалзиция целочисленной переменной classNum | 4 |
| 4 |  | ввод значения в переменную parentPath | 5 |
| 5 |  | вызов метода setName с параметром parentPath | 6 |
| 6 |  | ввод значения в переменную parentPath | 7 |
| 7 | paretnPath == endtree |  | ∅ |
|  | ввод значений в переменные childName и classNum | 8 |
| 8 |  | присвоение знаяения в переменную tmp полученного вызовом функции findByPath с параметров pathName | 9 |
| 9 | tmp не равно nullptr | Вывод "Object tree"  Запуск функции printTree  Вывод "\nThe head object "  Вывод значения переменной parentPath  Вывод " is not found" | ∅ |
|  |  | 10 |
| 10 | classNum == 2 | создание объекта в диначической памяти класса Two с параметрами tmp и childName | 15 |
|  |  | 11 |
| 11 | classNum == 3 | создание объекта в диначической памяти класса Three с параметрами tmp и childName | 15 |
|  |  | 12 |
| 12 | classNum == 4 | создание объекта в диначической памяти класса Four с параметрами tmp и childName | 15 |
|  |  | 13 |
| 13 | classNum == 5 | создание объекта в диначической памяти класса Five с параметрами tmp и childName | 15 |
|  |  | 14 |
| 14 | classNum == 6 | создание объекта в диначической памяти класса Six с параметрами tmp и childName | 15 |
|  |  | 15 |
| 15 |  |  | 6 |

## Алгоритм метода execute класса Application

Функционал: запуск системы объектов, установка и поиск объектов.

Параметры: -.

Возвращаемое значение: 0 - показатель успешного выполнения программы.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода execute класса Application

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Вывод строки "Object tree" | 2 |
| 2 |  | Вызов метода printTree | 3 |
| 3 |  | Иницилаизация строк path и command | 4 |
| 4 |  | Инициализация указателя setted = указателю на текущий объект | 5 |
| 5 |  | Ввод значения в переменную command | 6 |
| 6 | значение command != END | Вывод переноса строки | 7 |
|  |  | 7 |
| 7 | значение command != END |  | 8 |
|  |  | ∅ |
| 8 | значение command == SET | Ввод значения в переменную path | 9 |
|  |  | 11 |
| 9 | значение полученное вызовом метода findByPath() с параметром path != nullptr | присвоение в переменную setted значения полученного вызовом метода findByPath() с параметром path | 10 |
|  | Вывод сктроки "Object is not found  Вывод имени объекта под указателем setted  Вывод значения переменной path | 13 |
| 10 |  | Вывод строки "Object is set: "  Вывод имени объекта под указателем setted | 13 |
| 11 | значение command == "FIND" | Ввод значения в переменную path | 12 |
|  |  | 14 |
| 12 | значение полученное вызовом метода findByPath() с параметром path от объекта setted != nullptr | Вывод сктроки " Object name: "  Вывод имени объекта под указателем полученным вызовом метода findByPath() с параметром path от объекта setted | 13 |
|  | Вывод строки " Object is not found" | 13 |
| 13 |  | Ввод значение в переменную command | 14 |
| 14 | значение command!="END" | Вывод переноса строки | 7 |
|  |  | 7 |

## Алгоритм функции main

Функционал: точка входа.

Параметры: -.

Возвращаемое значение: код ошибки.

Алгоритм функции представлен в таблице 5.

Таблица 5 – Алгоритм функции main

| № | Предикат | Действия | № перехода |
| --- | --- | --- | --- |
| 1 |  | Создание объкекта root класса Аpplicatin с параметором nullptr | 2 |
| 2 |  | Вызов метода buildTree() для объекта root | 3 |
| 3 |  | Вернуть значение вызова метода execute() для объекта root | ∅ |

# 4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-12.

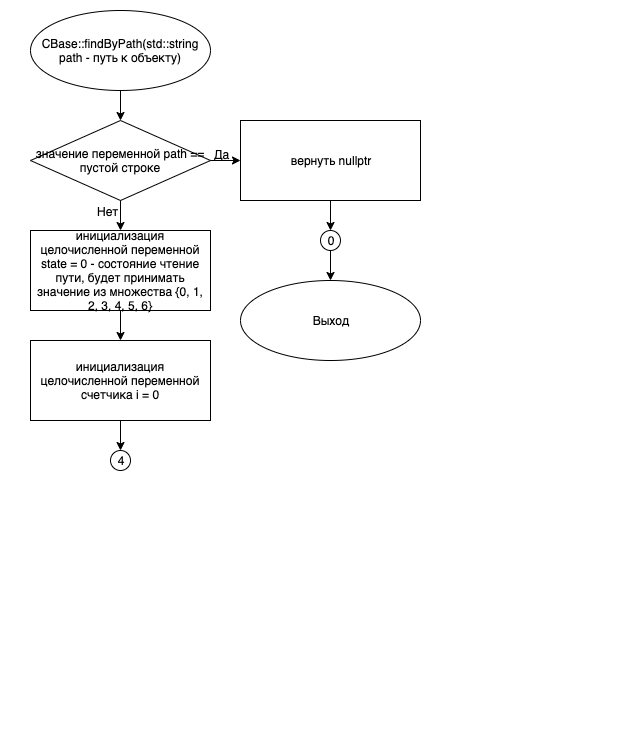


Рисунок 1 – Блок-схема алгоритма

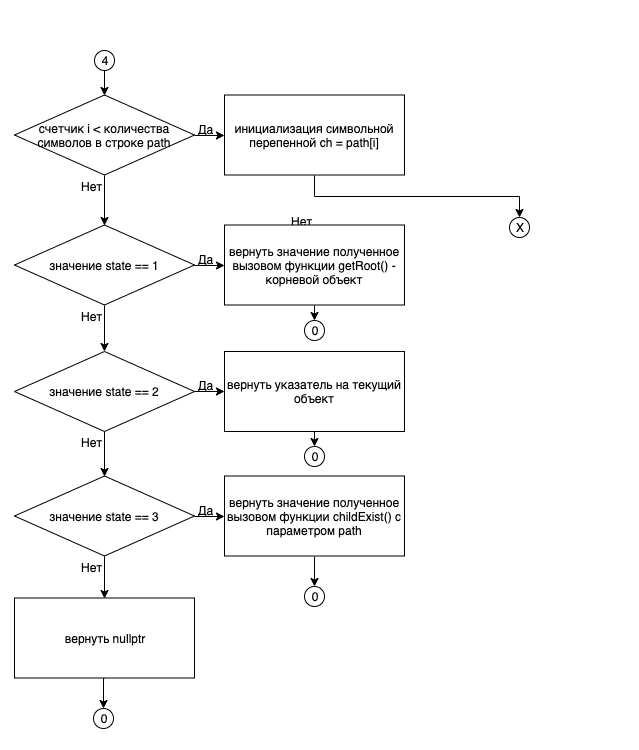


Рисунок 2 – Блок-схема алгоритма

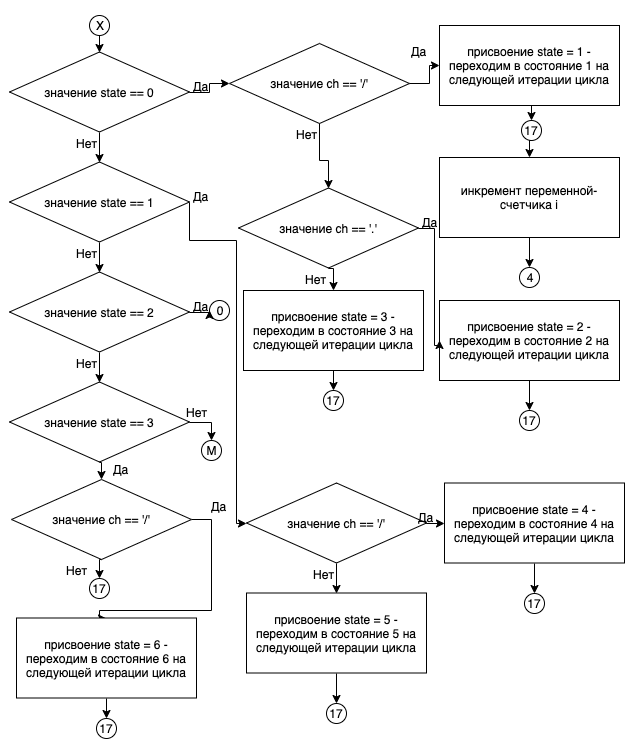


Рисунок 3 – Блок-схема алгоритма

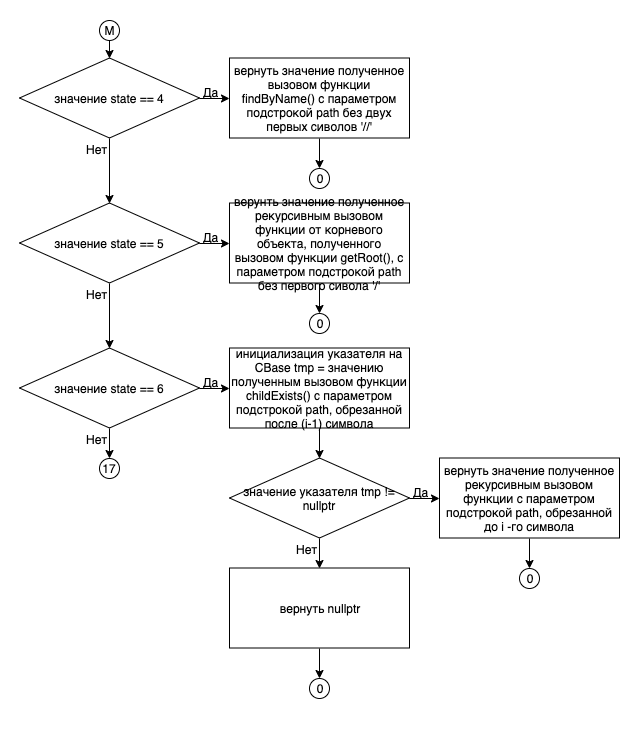


Рисунок 4 – Блок-схема алгоритма

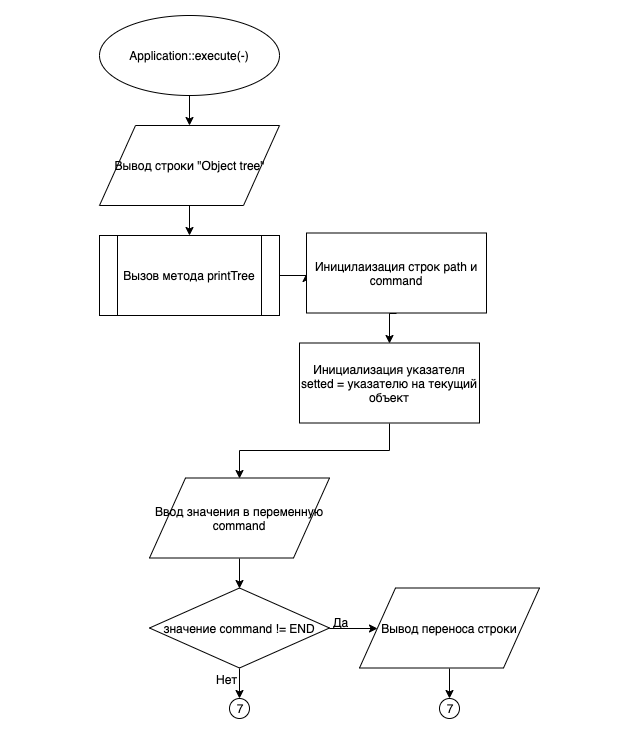


Рисунок 5 – Блок-схема алгоритма

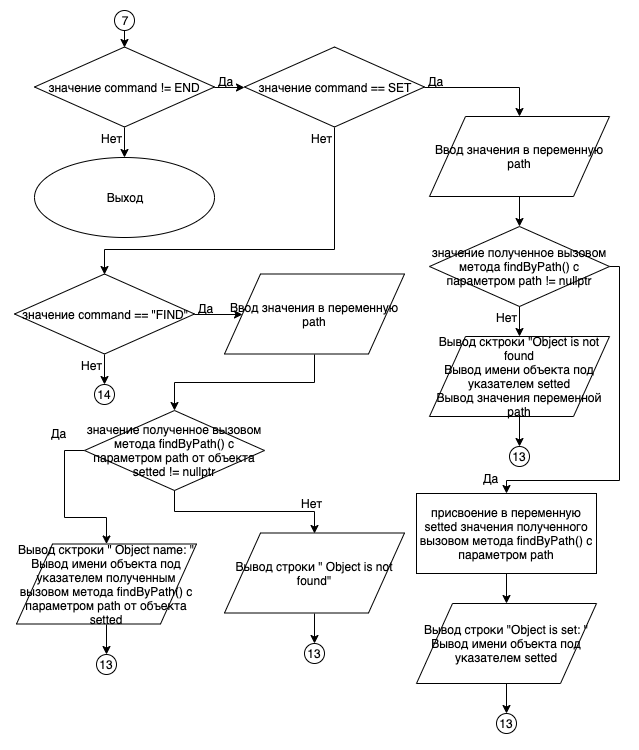


Рисунок 6 – Блок-схема алгоритма

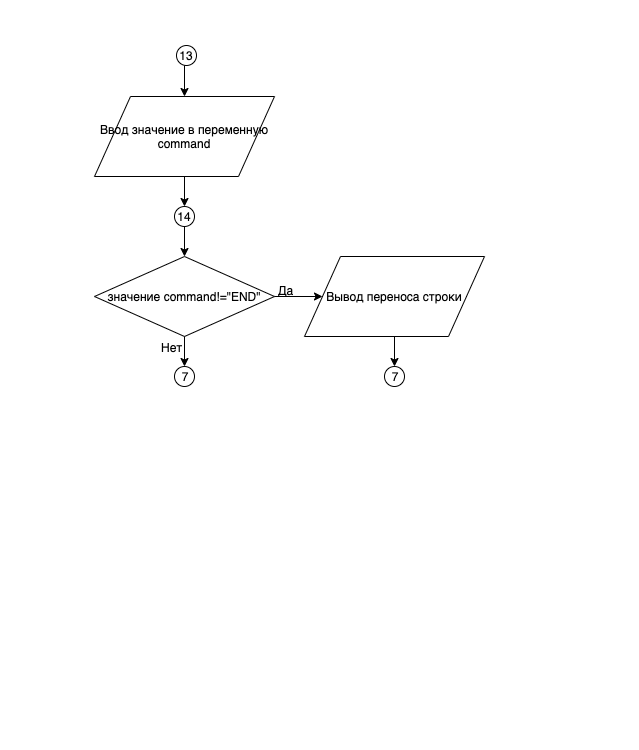


Рисунок 7 – Блок-схема алгоритма

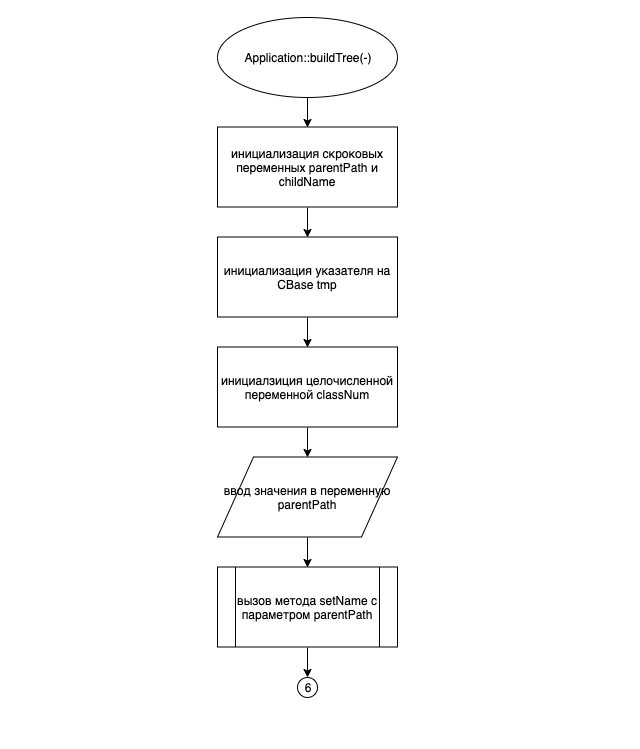


Рисунок 8 – Блок-схема алгоритма

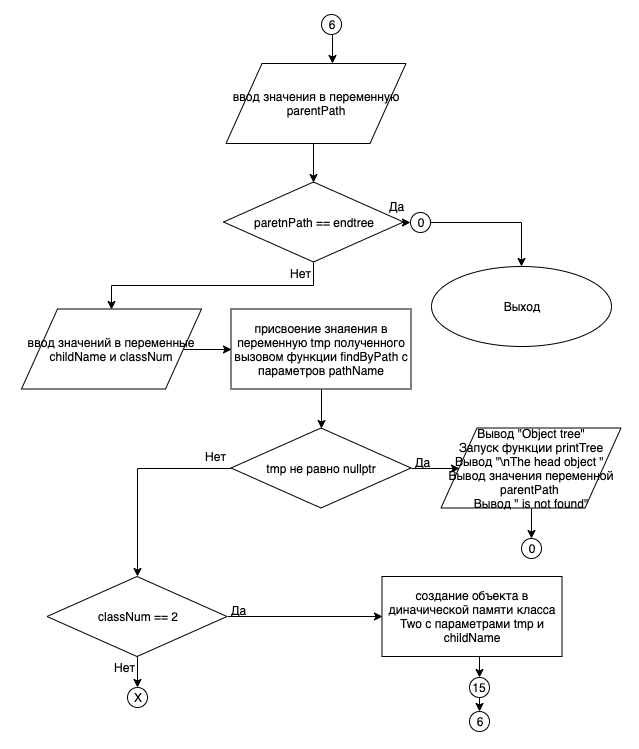


Рисунок 9 – Блок-схема алгоритма

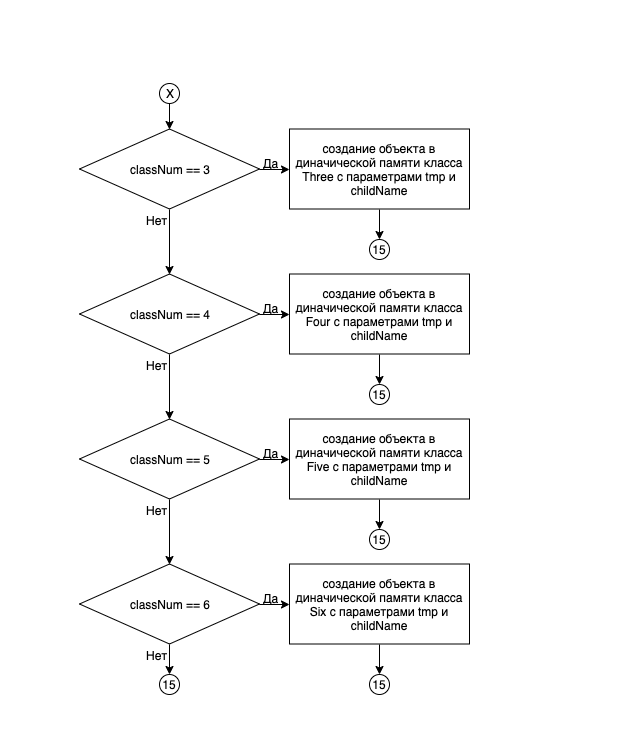


Рисунок 10 – Блок-схема алгоритма

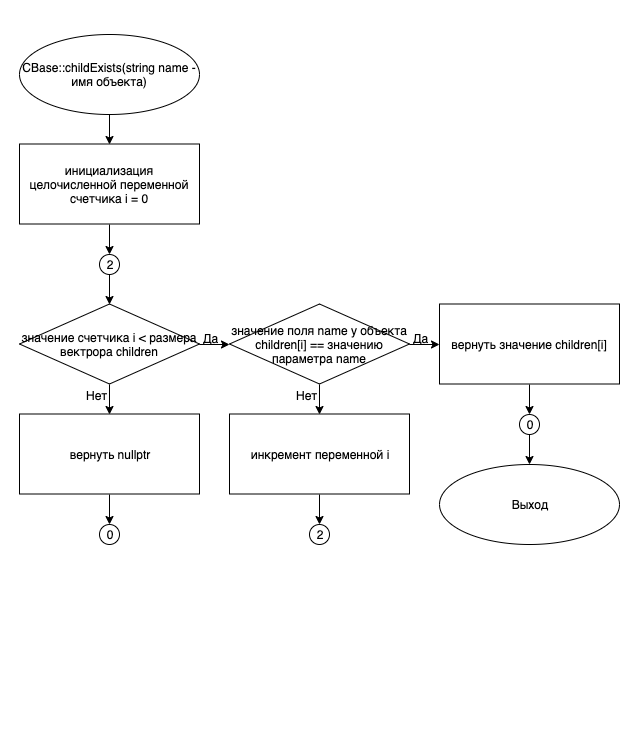


Рисунок 11 – Блок-схема алгоритма

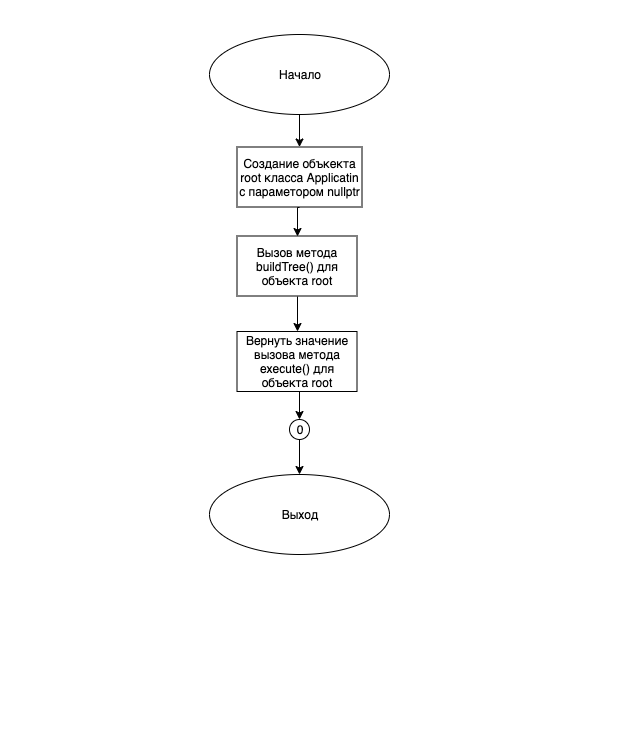


Рисунок 12 – Блок-схема алгоритма

# 5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

## Файл application.cpp

Листинг 1 – application.cpp

|  |
| --- |
| #include "application.h"  Application::Application(CBase\* parent, std::string name): CBase(parent, name){}  void Application::buildTree() {  std::string parentPath, childName;  CBase\* tmp;  int classNum;  std::cin >> parentPath;  this->setName(parentPath);  while(true) {  std::cin >> parentPath;  if (parentPath == "endtree") return;  std::cin >> childName >> classNum;  tmp = this->findByPath(parentPath);  if(!tmp) {  std::cout << "Object tree" << '\n';  this->printTree();  std::cout << "\nThe head object " << parentPath << " is not found";  exit(1);  }  switch(classNum) {  case 2:  new Two(tmp, childName);  break;  case 3:  new Three(tmp, childName);  break;  case 4:  new Four(tmp, childName);  break;  case 5:  new Five(tmp, childName);  break;  case 6:  new Six(tmp, childName);  break;  default:  break;  }  }  }  int Application::execute() {  std::cout << "Object tree" << '\n';  this->printTree();  std::string command, path;  CBase\* setted = this;  std::cin >> command;  if(command != "END")  std::cout << '\n';  while(command != "END") {  if(command == "SET") {  std::cin >> path;  if(setted->findByPath(path)) {  setted = setted->findByPath(path);  std::cout << "Object is set: "<<setted->getName();  }  else {  std::cout<<"Object is not found: "<< setted->getName()<<" "<<path;  }  }  if(command == "FIND") {  std::cin >> path;  if(setted->findByPath(path))  std::cout << path << " Object name: " << setted->findByPath(path)->getName();  else std::cout << path << " Object is not found";  }  std::cin >> command;  if(command!="END") std::cout << '\n';  }  return 0;  } |

## Файл application.h

Листинг 2 – application.h

|  |
| --- |
| #ifndef APPLICATION\_H  #define APPLICATION\_H  #include <iostream>  #include "cobject.h"  #include "cbase.h"  #include "two.h"  #include "three.h"  #include "four.h"  #include "five.h"  #include "six.h"  class Application: public CBase {  public:  Application(CBase\* parent, std::string name="Application");  void buildTree();  int execute();  };  #endif // APPLICATION\_H |

## Файл cbase.cpp

Листинг 3 – cbase.cpp

|  |
| --- |
| #include "cbase.h"  #include <iostream>  CBase::CBase(CBase\* parent, std::string name): parent(parent), name(name), readiness(0) {  if(parent != nullptr )  this->parent->children.push\_back(this);  }  void CBase::setName(std::string name) {  this->name = name;  }  std::string CBase::getName() {  return this->name;  }  void CBase::setReadiness(int readiness) {  if(readiness == 0) {  this->readiness = readiness;  for(CBase\* child : children)  child->setReadiness(readiness);  }  else {  CBase\* root = this;  while (root->parent != nullptr) {  if (root->parent->readiness == 0) {  this->readiness = 0;  return;  }  root = root-> parent;  }  this->readiness = readiness;  }  }  CBase\* CBase::findByName(std::string name) {  if (this->name == name) return this;  CBase\* root = this;  while(root->parent != nullptr) {  if(root->parent->name == name) return root->parent;  root = root->parent;  }  return root->getChild(name);  }  CBase\* CBase::findByPath(std::string path) {  if(path == "") return nullptr;  int state = 0; // состояние чтения пути  /\*  state == 0 - считывается первый символ  state == 1 - первый символ '/' - корневой объект  state == 2 - первый символ '.' - текущий объект  state == 3 - первый символ не '/', и не '.'  state == 4 - если первые два символа "//" - поиск объекта по уникальному имени от корневого  state == 5 - если первый символ '/', а следующий не '/' - абсолютная координата от корневого объекта  state == 6 - если встретилось '/' после имени объекта - относительная координата от текущего  \*/  for(size\_t i=0; i<path.size(); ++i) {  char ch = path[i];  switch(state) {  case 0: {  if(ch == '/') state = 1;  else if(ch == '.') state = 2;  else state = 3;  break;  }  case 1: {  if(ch == '/') state = 4;  else state = 5;  break;  }  case 2: {  return nullptr;  }  case 3: {  if(ch == '/') state = 6;  break;  }  case 4: {  return findByName(path.substr(2));  }  case 5: {  return getRoot()->findByPath(path.substr(1));  }  case 6: {  CBase\* tmp = this->childExists(path.substr(0, i-1));  if(tmp) {  return tmp->findByPath(path.substr(i));  }  return nullptr;  }  }  }  if(state == 1)  return getRoot();  if(state == 2)  return this;  if(state == 3)  return childExists(path);  return nullptr;  }  CBase\* CBase::childExists(std::string name) {  for(size\_t i=0; i<children.size(); ++i) {  if(children[i]->name == name){  return children[i];  }  }  return nullptr;  }  CBase\* CBase::getChild(std::string name) {  for(CBase\* child : children)  if(child->name == name) return child;  for(CBase\* child : children) {  if((child->children).size() > 0) {  CBase\* tempObj = child->getChild(name);  if (tempObj != nullptr)  return tempObj;  }  }  return nullptr;  }  CBase\* CBase::getRoot() {  CBase\* root = this;  while (root->parent != nullptr)  root = root->parent;  return root;  }  void CBase::setParent(CBase\* newParent) {  CBase\* oldParent = this->parent;  if (oldParent == nullptr || newParent == nullptr) return;  std::vector<CBase\*> oldPatherChildren = oldParent->children;  for (size\_t i=0; i < oldPatherChildren.size(); ++i)  if(oldPatherChildren[i] == this) {  oldPatherChildren.erase(oldPatherChildren.begin()+i);  break;  }  this->parent = newParent;  parent->children.push\_back(this);  }  void CBase::printTree(int level, bool readiness) {  if (parent == nullptr) {  std::cout << name;  std::cout << (readiness ? ((bool)(this->readiness) ? " is ready" : " is not ready") : "");  }  for (CBase\* child : children) {  std::cout << '\n' << std::string(4\*level, ' ') << child->name;  std::cout << (readiness ? ((bool)child->readiness ? " is ready" : " is not ready") : "");  child->printTree(level +1, readiness);  }  }  void CBase::printTreeWithReadiness() {  this->printTree(1, true);  }  CBase::~CBase() {  for(size\_t i=0; i < children.size(); ++i)  delete children[i];  } |

## Файл cbase.h

Листинг 4 – cbase.h

|  |
| --- |
| #ifndef CBASE\_H  #define CBASE\_H  #include <string>  #include <vector>  class CBase {  private:  CBase\* parent;  std::string name;  int readiness;  std::vector<CBase\*> children;  CBase\* getRoot();  public:  CBase(CBase\* parent, std::string name="Base");  void setName(std::string name);  std::string getName();  void setReadiness(int readiness);  void setParent(CBase\* parent);  CBase\* getParent();  CBase\* getChild(std::string name);  CBase\* findByName(std::string name);  CBase\* childExists(std::string name);  CBase\* findByPath(std::string path);  void printTree(int level=1, bool readiness=false);  void printTreeWithReadiness();  ~CBase();  };  #endif // CBASE\_H |

## Файл cobject.cpp

Листинг 5 – cobject.cpp

|  |
| --- |
| #include "cobject.h"  Cobject::Cobject(CBase\* parent, std::string name): CBase(parent, name){} |

## Файл cobject.h

Листинг 6 – cobject.h

|  |
| --- |
| #ifndef COBJECT\_H  #define COBJECT\_H  #include "cbase.h"  class Cobject: public CBase {  public:  Cobject(CBase\* parent, std::string name);  };  #endif // COBJECT\_H |

## Файл five.cpp

Листинг 7 – five.cpp

|  |
| --- |
| #include "five.h"  Five::Five(CBase\* parent, std::string name): CBase(parent, name){} |

## Файл five.h

Листинг 8 – five.h

|  |
| --- |
| #ifndef FIVE\_H  #define FIVE\_H  #include "cbase.h"  class Five: public CBase {  public:  Five(CBase\* parent, std::string name);  };  #endif // FIVE\_H |

## Файл four.cpp

Листинг 9 – four.cpp

|  |
| --- |
| #include "four.h"  Four::Four(CBase\* parent, std::string name): CBase(parent, name){} |

## Файл four.h

Листинг 10 – four.h

|  |
| --- |
| #ifndef FOUR\_H  #define FOUR\_H  #include "cbase.h"  class Four: public CBase {  public:  Four(CBase\* parent, std::string name);  };  #endif // FOUR\_H |

## Файл main.cpp

Листинг 11 – main.cpp

|  |
| --- |
| #include "application.h"  int main() {  Application root(nullptr);  root.buildTree();  return root.execute();  } |

## Файл six.cpp

Листинг 12 – six.cpp

|  |
| --- |
| #include "six.h"  Six::Six(CBase\* parent, std::string name): CBase(parent, name){} |

## Файл six.h

Листинг 13 – six.h

|  |
| --- |
| #ifndef SIX\_H  #define SIX\_H  #include "cbase.h"  class Six: public CBase {  public:  Six(CBase\* parent, std::string name);  };  #endif // SIX\_H |

## Файл three.cpp

Листинг 14 – three.cpp

|  |
| --- |
| #include "three.h"  Three::Three(CBase\* parent, std::string name): CBase(parent, name){} |

## Файл three.h

Листинг 15 – three.h

|  |
| --- |
| #ifndef THREE\_H  #define THREE\_H  #include "cbase.h"  class Three: public CBase {  public:  Three(CBase\* parent, std::string name);  };  #endif // THREE\_H |

## Файл two.cpp

Листинг 16 – two.cpp

|  |
| --- |
| #include "two.h"  Two::Two(CBase\* parent, std::string name): CBase(parent, name){}; |

## Файл two.h

Листинг 17 – two.h

|  |
| --- |
| #ifndef TWO\_H  #define TWO\_H  #include "cbase.h"  class Two: public CBase {  public:  Two(CBase\* parent, std::string name);  };  #endif // TWO\_H |

# 6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 6.

Таблица 6 – Результат тестирования программы

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

# ЗАКЛЮЧЕНИЕ

Таким образом, в результате проделанной работы была реализованна программа-система, которая строит иерархическое дерево и организовывает координирование его объектов.

Если же говорить о собственной пользе - благодаря работе с задачами и с самой системой "Аврора" в целом, я научился планировать и решать поставленные цели корректно и поэтапно. Этому поспособствовало четкое разделение по задачам внутри самой "среды". Я научился менее затратно в плане времени и сил, получать нужные мне результаты.

Также можно отметить удобство системы "Аврора" для создания подобных приложений, в которой реализована удобная система отображения ошибок при написании кода, а также система разработки метода и алгоритма решения.

# 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).