

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_bookmark0)

1. [ПОСТАНОВКА ЗАДАЧИ 7](#_bookmark1)
   1. [Описание входных данных 8](#_bookmark2)
   2. [Описание выходных данных 10](#_bookmark3)
2. [МЕТОД РЕШЕНИЯ 12](#_bookmark4)
3. [ОПИСАНИЕ АЛГОРИТМОВ 15](#_bookmark5)
   1. [Алгоритм метода findObjByPath класса Base 15](#_bookmark6)
   2. [Алгоритм метода buildTree класса App 17](#_bookmark7)
   3. [Алгоритм метода execute класса App 19](#_bookmark8)
   4. [Алгоритм функции main 21](#_bookmark9)
4. [БЛОК-СХЕМЫ АЛГОРИТМОВ 22](#_bookmark10)
5. [КОД ПРОГРАММЫ 26](#_bookmark11)
   1. [Файл App.cpp 26](#_bookmark12)
   2. [Файл App.h 27](#_bookmark13)
   3. [Файл Base.cpp 28](#_bookmark14)
   4. [Файл Base.h 30](#_bookmark15)
   5. [Файл Fifth.cpp 31](#_bookmark16)
   6. [Файл Fifth.h 31](#_bookmark17)
   7. [Файл Fourth.cpp 31](#_bookmark18)
   8. [Файл Fourth.h 32](#_bookmark19)
   9. [Файл main.cpp 32](#_bookmark20)

5.10 [Файл Object.cpp 32](#_bookmark21)

* 1. [Файл Object.h 32](#_bookmark22)
  2. [Файл Second.cpp 33](#_bookmark23)
  3. [Файл Second.h 33](#_bookmark24)
  4. [Файл Sixth.cpp 33](#_bookmark25)
  5. [Файл Sixth.h 34](#_bookmark26)
  6. [Файл Third.cpp 34](#_bookmark27)
  7. [Файл Third.h 34](#_bookmark28)

6 [ТЕСТИРОВАНИЕ 35](#_bookmark29)

[ЗАКЛЮЧЕНИЕ 36](#_bookmark31)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_bookmark32)

# ВВЕДЕНИЕ

В заключительной задаче курсовой работы необходимо использовать знания, полученнный за прохождение курса Объектно-ориентированного программирования, а именно:

взаимодействие системы объектов при помощи сигналов и обработчиков [5]; построение дерева иерархии [5, 6];

проектирование системы[5].

А также работа с техническим заданием, грамотное составление технической документации (а именно: написание метода решения, описание алгоритмов и оформление блок-схем согласно требованию приложения к методическому пособию[4])

Основываясь на методе решения, алгоритме и блок-схемах реализовать поставленную задачу, испоьзуя следующие парадигмы объектно- ориентированного программирования:

1. Инкапсуляция - механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то и другое от внешнего вмешательства или неправильного использования[2, c. 11];
2. Наследование - механизм, который позволяет одним классам включать в себя поля и методы других классов[1, с. 106].

# 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. **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. **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

# МЕТОД РЕШЕНИЯ

Объекты потокового ввода-вывода cin и cout Объект AppObj класса App

Объекты классов Second, Third, Fourth, Fifth,Sixth количество которых определяется пользовательским вводом

Класс Base, реализованный в работе 4\_1\_1 с дополнениями, добавленными в работе КЛ\_3\_1

Класс App, реализованный в работе 4\_1\_1 с дополнениями: Функционал

* Метод buildTree - строит иерархию объектов, состоящую из объектов

классов Second, Third, Fourth, Fifth, Sixth

* Метод execute - вызывает метод вывода дерева иерархии объектов, после чего выполняет введенные пользователем команды.

*Таблица 1 – Иерархия наследования классов*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Имя класса | Классы-  наследник и | Модифика  тор доступа | Описание | Номер | Комментар ий |
| 1 | Base |  |  | Базовый класс иерархии объектов содержит основные поля и  методы |  |  |
|  | App | public |  | 2 |  |
|  | Second | public |  | 3 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Third | public |  | 4 |  |
|  | Fourth | public |  | 5 |  |
|  | Fifth | public |  | 6 |  |
|  | Sixth | public |  | 7 |  |
| 2 | App |  |  | Класс приложени я - строит дерево объектов, задает готовность объектов дерева и выводит иерархию  объектов |  |  |
| 3 | Second |  |  | Класс объектов, подчиненн ых базовому классу  Base |  |  |
| 4 | Third |  |  | Класс объектов, подчиненн ых базовому классу  Base |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | Fourth |  |  | Класс объектов, подчиненн ых базовому классу  Base |  |  |
| 6 | Fifth |  |  | Класс объектов, подчиненн ых базовому классу  Base |  |  |
| 7 | Sixth |  |  | Класс объектов, подчиненн ых базовому классу  Base |  |  |

# ОПИСАНИЕ АЛГОРИТМОВ

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

* 1. **Алгоритм метода findObjByPath класса Base**

Функционал: возвращает указатель на искомый элемент иерархии (если такой существует), иначе возвращает нулевой указатель.

Параметры: string path - абсолютный путь к искомому объекту иерархии. Возвращаемое значение: Base\* - указатель на искомый элемент.

Алгоритм метода представлен в таблице 2.

*Таблица 2 – Алгоритм метода findObjByPath класса Base*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 | абсолютный путь = "." | Возвращение указателя на данный элемент | ∅ |
|  |  | 2 |
| 2 |  | Инициализация указателей на объект класса Base  \*currObj = this и \*root = this | 3 |
| 3 | существует родительский  объект | Присвоение указателю на корневой объект \*root  указателя на его родительский объект | 3 |
|  |  | 4 |
| 4 | абсолютный путь равен "/" | Возвращение указателя root на ранее найденный  корневой объект | ∅ |
|  |  | 5 |
| 5 | первые два символа абсолютного пути равны "/" | Возвращение значеия, возвращаемого от метода FindObjByName() вызванного у ранее найденного корневого объекта с подстрокой path без первых  двух символов | ∅ |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
|  |  |  | 6 |
| 6 | первый символ абсолютного пути равен "/", при этом, абсолютный путь указывает  на уровень ниже корневого | Присвоение строке абсолютного пути значения своей подстроки исключая первый символ | 7 |
|  |  | 7 |
| 7 |  | Объявление указателя foundObj на объект класса  Base | 8 |
| 8 |  | Инициализация строковой переменной name = "" | 9 |
| 9 |  | Инициализация целочисленной переменной i = 0 | 10 |
| 10 | значение i < длина строки  текущего абсолютного пути |  | 11 |
|  |  | 20 |
| 11 | i-тый символ строки абсолютного пути = '/' или i  = длина строки абсолютного  пути | Присвоение указателю foundObj значения нулевого указателя | 12 |
|  | Добавление в строку name i-того символа строки  абсолютного пути | 19 |
| 12 |  | Инициализация целочисленной переменной j = 0 | 13 |
| 13 | значение переменной j меньше кол-ва дочерних объектов объекта, на  который указывает currObj |  | 14 |
|  |  | 16 |
| 14 | имя j-того дочернего элемента объекта, на  который указывает currObj = | foundObj присваиваем указатель на j-тый дочерний объект, объекта, на который указывает currObj | 16 |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
|  | строка name |  |  |
|  |  | 15 |
| 15 |  | Увеличение значения переменной j на 1 | 13 |
| 16 |  | Присвоение currObj значения foundObj | 17 |
| 17 | указатель currObj так и не был изменен или по данному пути не был найден объект  иерархии | Возвращение нулевого указателя | ∅ |
|  |  | 18 |
| 18 |  | Присвоение name значения пустой строки | 19 |
| 19 |  | Увеличение значения переменной i на 1 | 10 |
| 20 |  | Возвращение указателя currObj | ∅ |

* 1. **Алгоритм метода buildTree класса App**

Функционал: строит дерево иерархии объектов по их абсолютному пути при помощи объектов классов Second, Third, Fourth, Fifth, Sixth.

Параметры: -. Возвращаемое значение: -.

Алгоритм метода представлен в таблице 3.

*Таблица 3 – Алгоритм метода buildTree класса App*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Объявление строковых перемен path и childName | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 2 |  | Ввод значения переменной path с клавиатуры | 3 |
| 3 |  | Вызов метода setName с параметром path | 4 |
| 4 |  | Объявление целочисленной переменной classNum | 5 |
| 5 |  | Объявление указателя parentObject на объект  класса Base | 6 |
| 6 | значение переменной path  вводится с клавиатуры |  | 7 |
|  |  | ∅ |
| 7 | значение строки path =  "endtree" |  | ∅ |
|  |  | 8 |
| 8 |  | Присвоение указателю parentObject значения, возвращаемого методом findObjByPath с  параметром path | 9 |
| 9 | Головной элемент не  существует | Вывод: "Object tree" | 10 |
|  |  | 13 |
| 10 |  | Вызов метода printHierarchyDepth() | 11 |
| 11 |  | Вывод с новой строки:"The head object (значение  переменной path)is not found" | 12 |
| 12 |  | Завершение работы программы вызовом функции  exit(0) | ∅ |
| 13 |  | Ввод значений переменных childName и classNum с  клавиатуры через разделитель | 14 |
| 14 | Значение переменной classNum = 2 | Создание объекта класса Second с параметрами parentObject и childName, передаваемыми в  конструктор | 6 |
|  |  | 15 |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 15 | Значение переменной classNum = 3 | Создание объекта класса Third с параметрами parentObject и childName, передаваемыми в  конструктор | 6 |
|  |  | 16 |
| 16 | Значение переменной classNum = 4 | Создание объекта класса Fourth с параметрами parentObject и childName, передаваемыми в  конструктор | 6 |
|  |  | 17 |
| 17 | Значение переменной classNum = 5 | Создание объекта класса Fifth с параметрами parentObject и childName, передаваемыми в  конструктор | 6 |
|  |  | 18 |
| 18 | Значение переменной classNum = 6 | Создание объекта класса Sixth с параметрами  parentObject и childName, передаваемыми в конструктор | 6 |
|  |  | 6 |

* 1. **Алгоритм метода execute класса App**

Функционал: выводит дерево иерархии объектов,после выполняет команды вводимые пользователем.

Параметры: -.

Возвращаемое значение: int, код ошибки работы приложения. Алгоритм метода представлен в таблице 4.

*Таблица 4 – Алгоритм метода execute класса App*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Вывод: "Object tree"  Вывод перехода на новую строку | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 2 |  | Вызов метода вывода иерархии объектов  printHierarchyDepth() | 3 |
| 3 |  | Инициализация указателя на объект класса Base currObj = указатель на текущий элемент Объявление указателя на объект класса Base  tmpObj | 4 |
| 4 |  | Объявление строковых переменных path и  command | 5 |
| 5 | значение переменной command вводится с  клавиатуры |  | 6 |
|  |  | 14 |
| 6 | значение переменной  command = "END" |  | 14 |
|  |  | 7 |
| 7 |  | Вывод: перевод на новую строку | 8 |
| 8 |  | Ввод значения path с клавиатуры | 9 |
| 9 |  | Присвоение указателю tmpObj значения,  возвращаемого методом findObjByPath, вызванным у currObj | 10 |
| 10 | значение переменной  command = "SET" |  | 11 |
|  |  | 13 |
| 11 | tmpObj != нулевому указателю или объект не был найден по пути path | Вывод: "Object is not found"(значение, возвращаемое методом getName(), вызванным у объекта currObj)"пробел"(значение переменной  path) | 5 |
|  |  | 12 |
| 12 |  | Присвоение указателю currObj значения указателя | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
|  |  | tmpObj  Вывод: "Object is set:"(значение, возвращаемое методом getName(), вызванным у объекта currObj) |  |
| 13 | tmpObj не равен нулевому указателю или объект не был  найден по пути path | Вывод:(значение переменной path)"4 пробела"Object is not found | 5 |
|  | Вывод: (значение переменной path)"4 пробела"  Object name:(значение, возвращаемое методом getName(), вызванным у tmpObj) | 5 |
| 14 |  | Возвращение значения метода 0 | ∅ |

* 1. **Алгоритм функции main**

Функционал: создает объект приложения, вызывает у него метод построения дерева иерархии и возвращает значение метода запуска приложения.

Параметры: -.

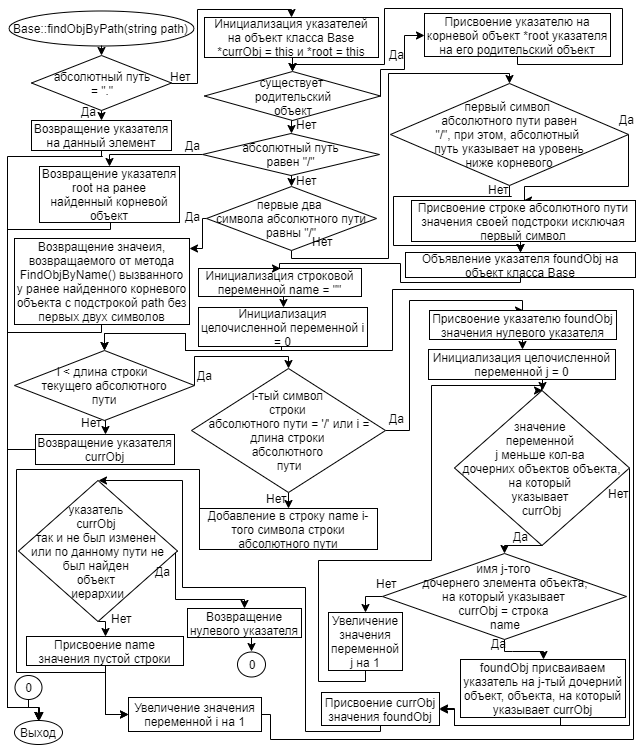
Возвращаемое значение: int, код ошибки. Алгоритм функции представлен в таблице 5.

*Таблица 5 – Алгоритм функции main*

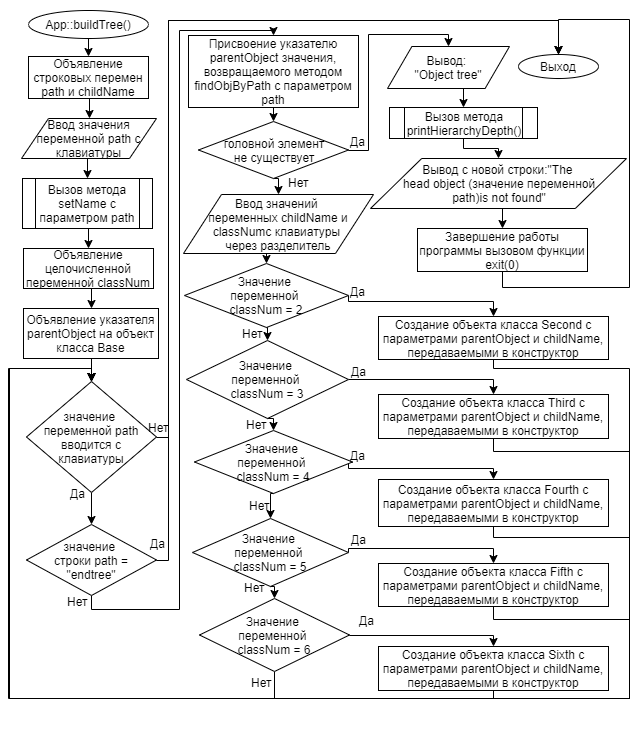
|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** |
| 1 |  | Создание объекта AppObj класса App с параметром nullptr | 2 |
| 2 |  | Вызов метода построения дерева иерархии у объекта AppObj | 3 |
| 3 |  | Возвращение значения метода execute(), вызываемого у объекта  AppObj | ∅ |

# БЛОК-СХЕМЫ АЛГОРИТМОВ

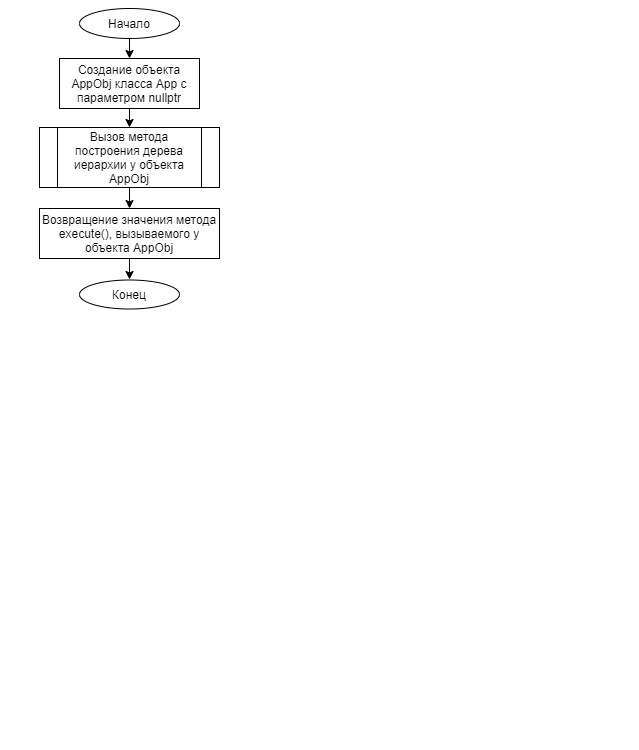
Представим описание алгоритмов в графическом виде на рисунках 1-4.



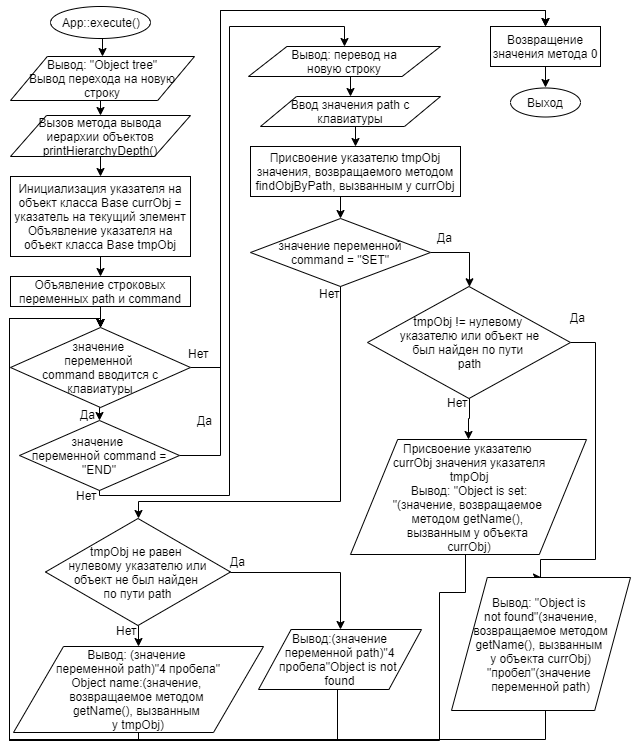
**Рисунок 1 – Блок-схема алгоритма**



**Рисунок 2 – Блок-схема алгоритма**



**Рисунок 3 – Блок-схема алгоритма**



**Рисунок 4 – Блок-схема алгоритма**

# КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена

ниже.

* 1. **Файл App.cpp**

*Листинг 1 – App.cpp*

#include <iostream> #include "App.h" #include "Second.h" #include "Third.h" #include "Fourth.h" #include "Fifth.h" #include "Sixth.h"

using namespace std;

App::App(Base\* parent): Base(parent){} void App::buildTree()

{

string path, childName; cin >> path; setName(path);

int classNum;

Base\* parentObject; while (cin >> path){

if (path == "endtree") break; parentObject = findObjByPath(path); if (!parentObject){

cout<<"Object tree"<< endl; printHierarchyDepth();

cout<<endl<<"The head object " << path << " is not found"; exit(0);

}

cin>>childName>>classNum; switch(classNum){

case 2:

new Second(parentObject, childName); break;

case 3:

new Third(parentObject, childName); break;

case 4:

new Fourth(parentObject, childName); break;

case 5:

new Fifth(parentObject, childName);

break; case 6:

new Sixth(parentObject, childName); break;

}

}

}

int App::execute(){

cout<<"Object tree"<< endl; printHierarchyDepth();

Base\* currObj = this, \*tmpObj; string path, command;

while (cin >> command){

if (command=="END") break; cout<<endl;

cin>>path;

tmpObj=currObj->findObjByPath(path); if (command=="SET"){

if(!tmpObj){

cout<<"Object is not found: "<<currObj->getName()<<"

"<<path;

}

else{

currObj = tmpObj;

cout<<"Object is set: "<<currObj->getName();

}

}

else{

if (!tmpObj){

cout<<path<<"

}

else{

cout<<path<<"

}

"<<"Object is not found";

"<<"Object name: "<< tmpObj->getName();

}

}

return 0;

}

* 1. **Файл App.h**

*Листинг 2 – App.h*

#ifndef APP\_H #define APP\_H #include "Base.h"

class App: public Base{ public:

App(Base\* parent); void buildTree(); int execute();

};

#endif

* 1. **Файл Base.cpp**

*Листинг 3 – Base.cpp*

#include <iostream> #include "Base.h"

using namespace std;

Base::Base(Base \* parent, string name){ this->name = name;

parentObj = parent;

if (parentObj) parentObj->children.push\_back(this);

}

void Base::setName(string name){ this->name = name;

}

string Base::getName(){ return name;

}

void Base::printHierarchy(){

if (!parentObj) cout << name; if (children.size()){

cout << endl << name;

for (int i = 0;i < children.size();i++){

cout << " " << children[i]->getName();

}

children[children.size()-1]->printHierarchy();

}

}

void Base::printHierarchyDepth(int depth){ cout<<name;

if (children.size()){ depth++;

for (int i = 0; i < children.size(); i++){ cout << endl;

for (int j = 0; j < depth; j++) cout << " children[i]->printHierarchyDepth(depth);

}

}

}

";

void Base::printHierarchyDepthState(int depth){ cout << name;

cout << ((state == 0) ? " is not ready" : " is ready"); if (children.size()){

depth++;

for (int i = 0; i < children.size();i++){ cout << endl;

for (int j = 0; j < depth; j++) cout << " "; children[i]->printHierarchyDepthState(depth);

}

}

}

void Base::setParent(Base\* newParent){ if (parentObj && newParent){

for (int i=0;i < parentObj->children.size();i++){ if (parentObj->children[i] == this){

parentObj->children.erase(parentObj-

>children.begin() + i);

}

}

break;

parentObj = newParent;

parentObj->children.push\_back(this);

}

}

Base\* Base::getParent(){ return parentObj;

}

Base::~Base(){

for (int i = 0; i < children.size(); i++) delete children[i];

}

int Base::getState(){ return state;

}

void Base::setState(int state){

if (parentObj && parentObj->getState() == 0) return; else{

if (state == 0){

for (int i = 0;i < children.size();i++) children[i]-

>setState(0);

}

this->state = state;

}

}

Base\* Base::findObjByName(string name){ Base\* currObj;

if (this->name == name)return this;

for (int i = 0;i < children.size();i++){

currObj = children[i]->findObjByName(name); if (currObj) return currObj;

}

return nullptr;

}

Base\* Base::findObjByPath(string path){

if (path==".") return this;

Base\* currObj = this, \*root = this;

while(root->getParent()) root = root->getParent(); if (path=="/"){

return root;

}

if (path[0]=='/' && path[1]=='/'){

return root->findObjByName(path.substr(2));

}

if (path[0]=='/')

path=path.substr(1);

Base\* foundObj; string name="";

for (int i = 0;i<=path.length();i++){ if(path[i]=='/' || i==path.length()){ foundObj = nullptr;

for(int j=0;j<currObj->children.size();j++){ if(currObj->children[j]->getName()==name){

foundObj=currObj->children[j]; break;

}

}

currObj=foundObj; if(!currObj) return nullptr; name="";

}

else name+=path[i];

}

return currObj;

}

* 1. **Файл Base.h**

*Листинг 4 – Base.h*

#ifndef BASE\_H #define BASE\_H #include <vector> #include <string>

using namespace std;

class Base{ private:

string name;

Base \* parentObj = nullptr; vector <Base\*> children; int state = 0;

public:

Base(Base \* parent, string name = ""); void setName(string name);

string getName();

void printHierarchy();

void setParent(Base\* newParent); Base\* getParent();

~Base();

int getState();

void setState(int state);

void printHierarchyDepth(int depth = 0);

void printHierarchyDepthState(int depth = 0); Base\* findObjByName(string name);

Base\* findObjByPath(string path);

};

#endif

* 1. **Файл Fifth.cpp**

*Листинг 5 – Fifth.cpp*

#include "Fifth.h"

Fifth::Fifth(Base\* parent, string name) : Base(parent, name){

}

* 1. **Файл Fifth.h**

*Листинг 6 – Fifth.h*

#ifndef FIFTH\_H #define FIFTH\_H #include "Base.h"

class Fifth : public Base{ public:

Fifth(Base\* parent, string name = "");

};

#endif

* 1. **Файл Fourth.cpp**

*Листинг 7 – Fourth.cpp*

#include "Fourth.h"

Fourth::Fourth(Base\* parent, string name) : Base(parent, name){

}

* 1. **Файл Fourth.h**

*Листинг 8 – Fourth.h*

#ifndef FOURTH\_H #define FOURTH\_H #include "Base.h"

class Fourth : public Base{ public:

Fourth(Base\* parent, string name = "");

};

#endif

* 1. **Файл main.cpp**

*Листинг 9 – main.cpp*

#include "App.h"

int main(){

App AppObj(nullptr); AppObj.buildTree(); return AppObj.execute();

}

* 1. **Файл Object.cpp**

*Листинг 10 – Object.cpp*

#include "Object.h"

Object::Object(Base\* parent, string name):Base(parent, name){}

* 1. **Файл Object.h**

*Листинг 11 – Object.h*

#ifndef OBJECT\_H #define OBJECT\_H #include "Base.h"

class Object : public Base

{

public:

Object(Base\* parent, string name = "");

};

#endif

* 1. **Файл Second.cpp**

*Листинг 12 – Second.cpp*

#include "Second.h"

Second::Second(Base\* parent, string name) : Base(parent, name){

}

* 1. **Файл Second.h**

*Листинг 13 – Second.h*

#ifndef SECOND\_H #define SECOND\_H #include "Base.h"

class Second : public Base{ public:

Second(Base\* parent, string name = "");

};

#endif

* 1. **Файл Sixth.cpp**

*Листинг 14 – Sixth.cpp*

#include "Sixth.h"

Sixth::Sixth(Base\* parent, string name) : Base(parent, name){

}

* 1. **Файл Sixth.h**

*Листинг 15 – Sixth.h*

#ifndef SIXTH\_H #define SIXTH\_H #include "Base.h"

class Sixth : public Base{ public:

Sixth(Base\* parent, string name = "");

};

#endif

* 1. **Файл Third.cpp**

*Листинг 16 – Third.cpp*

#include "Third.h"

Third::Third(Base\* parent, string name) : Base(parent, name){

}

* 1. **Файл Third.h**

*Листинг 17 – Third.h*

#ifndef THIRD\_H #define THIRD\_H #include "Base.h"

class Third : public Base{ public:

Third(Base\* parent, string name = "");

};

#endif

# ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице [6](#_bookmark30).

*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 |

# ЗАКЛЮЧЕНИЕ

При помощи полученных за курс Объектно-ориентированного программирования знаний реализовать поставленную задачу. Провести необходимые тестирования кода и убедиться в его работоспособности. За пройденный курс Объектно-ориентированного программирования приобрел такие знания, как:

понимание понятия класса "класс" и "объект", обрел базовые умения работы с объектами и классами[2];

узнал про перегрузку операторов, получил необходимые навыки работы с перегрузкой операторов[2];

узнал основные парадигмы Объектно-ориентированного программирования[2]; приобрел умение работать с дружествеенными функциями[2].

Также в ходе выполнения курсовой работы был получен незаменимый опыт проектировки системы и работы с документацией, а именно с техническим заданием.

Поставленная задача была успешно решена. Был описан метод решения, описан алгоритм всех необходимых функций и методов, отрисованы блок-схемы по всем описанным в алгоритме методам и функциям.

Разработка программного продукта проходила в учебно-технологической среде "ACO Avrora". Следует отметить, что данная среда разработки сильно упрощает процесс оформления работ, а именно:

1. "ACO Avrora" имеет удобный функционал для построения алгоритмов решаемой задачи;
2. "ACO Avrora" предоставляет возможность генерации блок-схем по алгоритму;
3. "ACO Avrora" позволяет проводить тестирование с автоматическим

сравнением выводимых программой данных с ожидаемыми;

1. "ACO Avrora" работает круглосуточно
2. "ACO Avrora" имеет возможность автоматической генерации отчета
3. "ACO Avrora" реализована система контроля версий (избавляет от необходимости тратить время на перенесение программы на другие носители для работы на разных устройствах).

# 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.

1. 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).
2. 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).
3. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».
4. 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).