

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Институт информационных технологий

Кафедра вычислительной техники

**КУРСОВАЯ РАБОТА**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| по дисциплине | | | | Объектно-ориентированное программирование | | |
|  |  |  |  | (наименование дисциплины) |  |  |
| **Тема курсовой работы** Разработка программы для поиска объекта на дереве иерархии по | | | | | | |
|  | |  |  |  |  |  |
| уникальному наименованию | | | |  |  |  |
| **Студент группы** | | | | |  |  |
|  |  |  |  | (ФИО, учебная группа) |  | (подпись студента) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Руководитель курсовой работы** | | | |  | |  | | |
|  |  |  |  | (должность, звание, ученая степень, ФИО) | | | | | | |  | (подпись руководителя) |
| **Рецензент** (при наличии) |  |  |  |  |  | |  |  | |  |  |  |
|  |  |  |  | (должность, звание, ученая степень, ФИО) | | | | | | |  | (подпись рецензента) |
| Работа представлена к защите | « |  |  |  | » | |  |  | | 2021г. |  |  |
| Допущен к защите | « |  |  |  | » | |  |  | | 2021г. |  |  |

Москва 2021 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Институт информационных технологий

Кафедра вычислительной техники

**Утверждаю**

Заведующий кафедрой

*Подпись*

Платонова О.В.

*ФИО*

«25» февраля 2021 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы** по дисциплине

«Объектно-ориентированное программирование»

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | Милованов Дмитрий Алексеевич | Группа ИКБО-04-20 | |
|  |  |  |  |
| **Тема:** | Разработка программы для поиска объекта на дереве иерархии по уникальному | | |
|  | наименованию |  |  |

**Исходные данные:** Исходная иерархия расположения объектов, координаты искомых объектов **Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

1. Реализовать алгоритм размещения всех объектов в составе программы на иерархическом дереве объектов.
2. Реализовать алгоритм поиска объекта на дереве иерархии.
3. Блок-схема реализованных алгоритмов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Срок представления к защите курсовой работы:** | | **до «31» мая 2021г.** | |  |  |
| **Задание на курсовую работу выдал** | | ( | |  | ) |
|  | *Подпись руководителя* |  |  | *Ф.И.О. руководителя* |  |
| **Задание на курсовую работу получил** | | «25» февраля 2021г | |  |  |
|  |  | ( | | . | ) |
|  | *Подпись обучающегося* |  |  | Ф.И.О. исполнителя |  |

Москва 2021 г.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 4

Постановка задачи 6

Метод решения 9

Описание алгоритма 11

Блок-схема алгоритма 17

Код программы 25

Тестирование 32

ЗАКЛЮЧЕНИЕ 33

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (ИСТОЧНИКОВ) 34

**ВВЕДЕНИЕ**

Объектно-ориентированное программирование (ООП) - парадигма программирования, которая основана на представлении программы в виде совокупности объектов.

Объект - некоторая сущность, у которой есть определенные состояния, свойства, операции, поведение. Каждый из объектов является экземпляром определенного класса, совокупность которых образует иерархию наследования.

ООП является одной из самых популярных парадигм программирования. Оно уже долгое время является актуальным и не собирается сбавлять ход.

* ООП есть множество понятий, как абстракция данных, инкапсуляция и наследование. Они позволяют упрощать код, деля их на более маленькие. Это делает код простым для понимания. Возможно использовать один и тот же код в разных проектах и не переписывать его в одном проекте много раз. Благодаря этому программисты могут сэкономить большое количество времени.

Также актуальность этой парадигмы связана с реальным миром. ООП очень похоже на наше окружение: объектами являются люди, животные, предметы, а методами - их действия.

ООП помогает развить объектное мышление. Этот навык позволяет быстрее и легче проектировать сложные системы, представляя их в виде объектов.

Важный аспект ООП состоит том, что его можно реализовать во многих языках программирования. Однако одним из лидеров является C++:

4

* в нем изначально реализованы элементы ООП: классы, методы и другие;
* четко разграничены уровни доступа к членам класса;
* C++ поощряет программистов свойствами вроде множественного наследования и динамического полиморфизма;
* C++ дает большую свободу программисту, что позволяет ему писать продвинутый и быстрый код.

На моем направлении C++ и ООП дают преимущества в дальнейшем обучении. Изучив их на первом курсе, впоследствии я смогу быстрее и качественнее писать программы на одном из самых производительных и многофункциональных языках программирования.

5

**Постановка задачи**

**Определение указателя на объект по его координате**

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

* составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов.
* качестве параметра методу передать путь объекта от корневого. Путь задать в следующем виде:

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

Уникальность наименования требуется только относительно множества подчиненных объектов для любого головного объекта.

Если система содержит объекты с уникальными именами, то в методе реализовать определение указателя на объект посредством задания координаты в виде:

//«наименование объекта»

Состав и иерархия объектов строится посредством ввода исходных данных.

Ввод организован как в контрольной работе № 1.

Единственное различие: в строке ввода первым указать не наименование головного объекта, а путь к головному объекту.

6

Подразумевается, что к моменту ввода очередной строки соответствующая ветка на дереве иерархии уже построена.

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

**Пример ввода иерархии дерева объектов.**

root

/root object\_1 3 1

/root object\_2 2 1

/root/object\_2 object\_4 3 -1

/root/object\_2 object\_5 4 1

/root object\_3 3 1

/root/object\_2 object\_3 6 1

/root/object\_1 object\_7 5 1

/root/object\_2/object\_4 object\_7 3 -1

endtree

**Описание входных данных**

Множество объектов, их характеристики и расположение на дереве иерархии.

Структура данных для ввода согласно изложенному в фрагменте методического указания [3] в контрольной работе № 1.

После ввода состава дерева иерархии построчно вводятся координаты искомых объектов.

Ввод завершается при вводе: //

**Описание выходных данных**

**Первая строка:**

7

Object tree

**Со второй строки** вывести иерархию построенного дерева.

**Далее, построчно**:

«координата объекта» Object name: «наименование объекта» Разделитель один пробел.

Если объект не найден, то вывести:

«координата объекта» Object not found

Разделитель один пробел.

8

**Метод решения**

Объект base класса BaseApplication.

Класс Base:

Методы:

* Метод get\_object\_by\_path(string path) - возвращает указатель на искомый объект в составе дерева иерархии объектов по пути или координате.

Класс BaseApplication:

Методы:

* Метод build\_tree\_objects() - строит исходное дерево иерархии объектов;
* Метод exec\_app() - запускает приложение.

Иерархия наследования отображена в таблице 1.

Таблица 1. "Описание иерархии наследования классов"

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Модификатор | |  |  |  |  |
| № |  | Имя класса | Классы-наследники | доступа | при | Описание | |  | Номер |
|  |  |  |  | наследовании | |  |  |  |  |
|  | 1 | Base |  |  |  | Базовый класс | | |  |
|  |  |  |  |  |  | в | иерархии | |  |
|  |  |  |  |  |  | классов. | |  |  |
|  |  |  |  |  |  | Содержит | |  |  |
|  |  |  |  |  |  | основные | |  |  |
|  |  |  |  |  |  | свойства | | и |  |
|  |  |  |  |  |  | методы. | |  |  |
|  |  |  | BaseApplication | public |  |  |  |  | 2 |
|  |  |  | Class2 | public |  |  |  |  | 3 |
|  |  |  | Class3 | public |  |  |  |  | 4 |

9

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Class4 | public |  |  | 5 |
|  |  | Class5 | public |  |  | 6 |
|  |  | Class6 | public |  |  | 7 |
| 2 | BaseApplication |  |  | Класс |  |  |
|  |  |  |  | корневого | |  |
|  |  |  |  | объекта | |  |
|  |  |  |  | (приложения). | |  |
| 3 | Class2 |  |  | Класс, | |  |
|  |  |  |  | соответствую | |  |
|  |  |  |  | щий номеру 2. | |  |
|  |  |  |  | Используется | |  |
|  |  |  |  | для | объектов |  |
|  |  |  |  | дерева. | |  |
| 4 | Class3 |  |  | Класс, | |  |
|  |  |  |  | соответствую | |  |
|  |  |  |  | щий номеру 3. | |  |
|  |  |  |  | Используется | |  |
|  |  |  |  | для | объектов |  |
|  |  |  |  | дерева. | |  |
| 5 | Class4 |  |  | Класс, | |  |
|  |  |  |  | соответствую | |  |
|  |  |  |  | щий номеру 4. | |  |
|  |  |  |  | Используется | |  |
|  |  |  |  | для | объектов |  |
|  |  |  |  | дерева. | |  |
| 6 | Class5 |  |  | Класс, | |  |
|  |  |  |  | соответствую | |  |
|  |  |  |  | щий номеру 5. | |  |
|  |  |  |  | Используется | |  |
|  |  |  |  | для | объектов |  |
|  |  |  |  | дерева. | |  |
| 7 | Class6 |  |  | Класс, | |  |
|  |  |  |  | соответствую | |  |
|  |  |  |  | щий номеру 6. | |  |
|  |  |  |  | Используется | |  |
|  |  |  |  | для | объектов |  |
|  |  |  |  | дерева. | |  |

10

**Описание алгоритма**

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

Класс объекта: Base

Модификатор доступа: public

Метод: get\_object\_by\_path

Функционал: возвращает указатель на искомый объект в составе дерева иерархии объектов по пути или координате

Параметры: string path - путь к искомому объекту или его координата

Возвращаемое значение: Base\*, указатель на искомый объект

Алгоритм метода представлен в таблице 2.

Таблица 2. Алгоритм метода get\_object\_by\_path класса Base

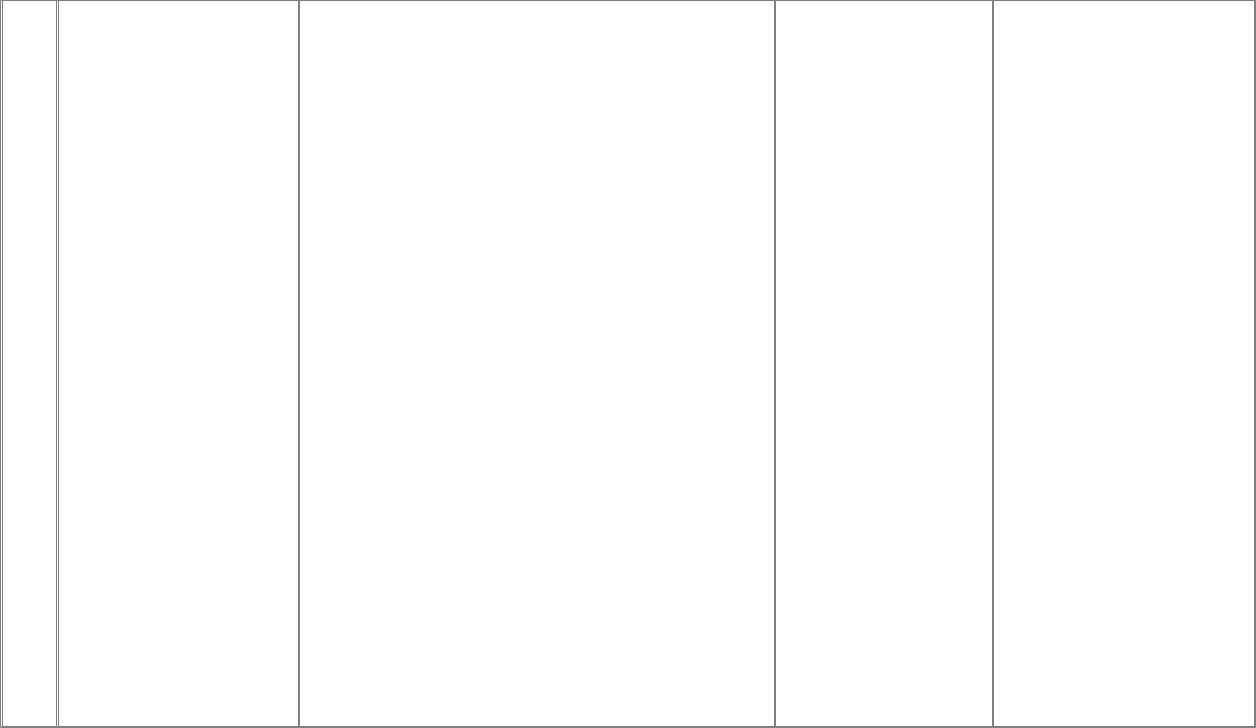
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** | **Комментарий** |  |
| 1 |  | Объявление строковой | 2 |  |  |
|  | переменной name |  |  |
|  |  |  |  |  |
|  |  | Присвоение переменной |  |  |  |
|  | Второй | name результата вызова |  |  |  |
| 2 | символ в path | метода substr переменной | 3 |  |  |
| - '/' | path с аргументами 2 и |  |  |  |
|  |  | "длина path" - 2 |  |  |  |
|  |  |  | 4 |  |  |
|  |  | Возврат результата вызова | ∅ |  |  |
| 3 |  | метода get\_object\_by\_name с |  |  |
|  |  | аргументом name |  |  |  |
| 4 |  | Инициализация | 5 |  |  |
|  |  | целочисленной переменной |  |  |  |

11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | start со значением 1 и end со |  |  |  |
|  |  | значением длины path |  |  |  |
|  |  | Инициализация |  |  |  |
| 5 |  | целочисленной переменной i | 6 |  |  |
|  |  | со значением 1 |  |  |  |
|  | i меньше |  | 7 |  |  |
| 6 | длины path |  |  |  |
|  |  |  |  |
|  |  |  | 10 |  |  |
|  | i-ый символ |  | 8 |  |  |
| 7 | path - '/' |  |  |  |
|  |  |  |  |
|  |  |  | 9 |  |  |
|  | start равен -1 | Присвоение переменной start | 9 |  |  |
| 8 | значения i |  |  |
|  | Присвоение переменной end | 10 |  |  |
|  |  |  |  |
|  |  | значения i |  |  |  |
| 9 |  | Увеличение значения i на 1 | 6 |  |  |
|  | start равен -1 | Возврат указателя на данный | ∅ |  |  |
|  |  | объект |  |  |  |
| 10 |  | Присвоение переменной |  |  |  |
|  | name результата вызова |  |  |  |
|  |  | метода substr переменной | 11 |  |  |
|  |  | path с аргументами start + 1 и |  |  |  |
|  |  | end - start - 1 |  |  |  |
|  |  | Объявление строковой |  |  |  |
| 11 |  | переменной | 12 |  |  |
|  |  | child\_object\_name |  |  |  |
|  |  | Инициализация указателя |  |  |  |
| 12 |  | child\_object на первый | 13 |  |  |
|  | объект в массиве |  |  |
|  |  |  |  |  |
|  |  | children\_object\_list |  |  |  |
|  |  | Присвоение переменной |  |  |  |
|  | child\_object не | child\_object\_name результата |  |  |  |
|  | равен | вызова метода | 14 |  |  |
| 13 | нулевому | get\_object\_name объекта, на |  |  |
|  | указателю | который указывает |  |  |  |
|  |  | child\_object |  |  |  |
|  |  |  | 17 |  |  |

12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Присвоение переменной |  |  |
|  | child\_object\_n | name результата вызова |  |  |
| 14 | ame равен | метода substr переменной | 15 |  |
| name | path с аргументами start и |  |  |
|  |  | "длина path" - start |  |  |
|  |  |  | 16 |  |
|  |  | Возврат результата вызова |  |  |
|  |  | метода get\_object\_by\_path | ∅ |  |
| 15 |  | объекта, на который |  |
|  |  | указывает child\_object, с |  |  |
|  |  | аргументом name |  |  |
|  |  | Направление указателя |  |  |
| 16 |  | child\_object на следующий | 13 |  |
|  | объект в массиве |  |
|  |  |  |  |
|  |  | children\_object\_list |  |  |
| 17 |  | Возврат нулевого указателя | ∅ |  |

Класс объекта: BaseApplication

Модификатор доступа: public

Метод: build\_tree\_objects

Функционал: строит исходное дерево иерархии объектов

Параметры: нет

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода build\_tree\_objects класса BaseApplication

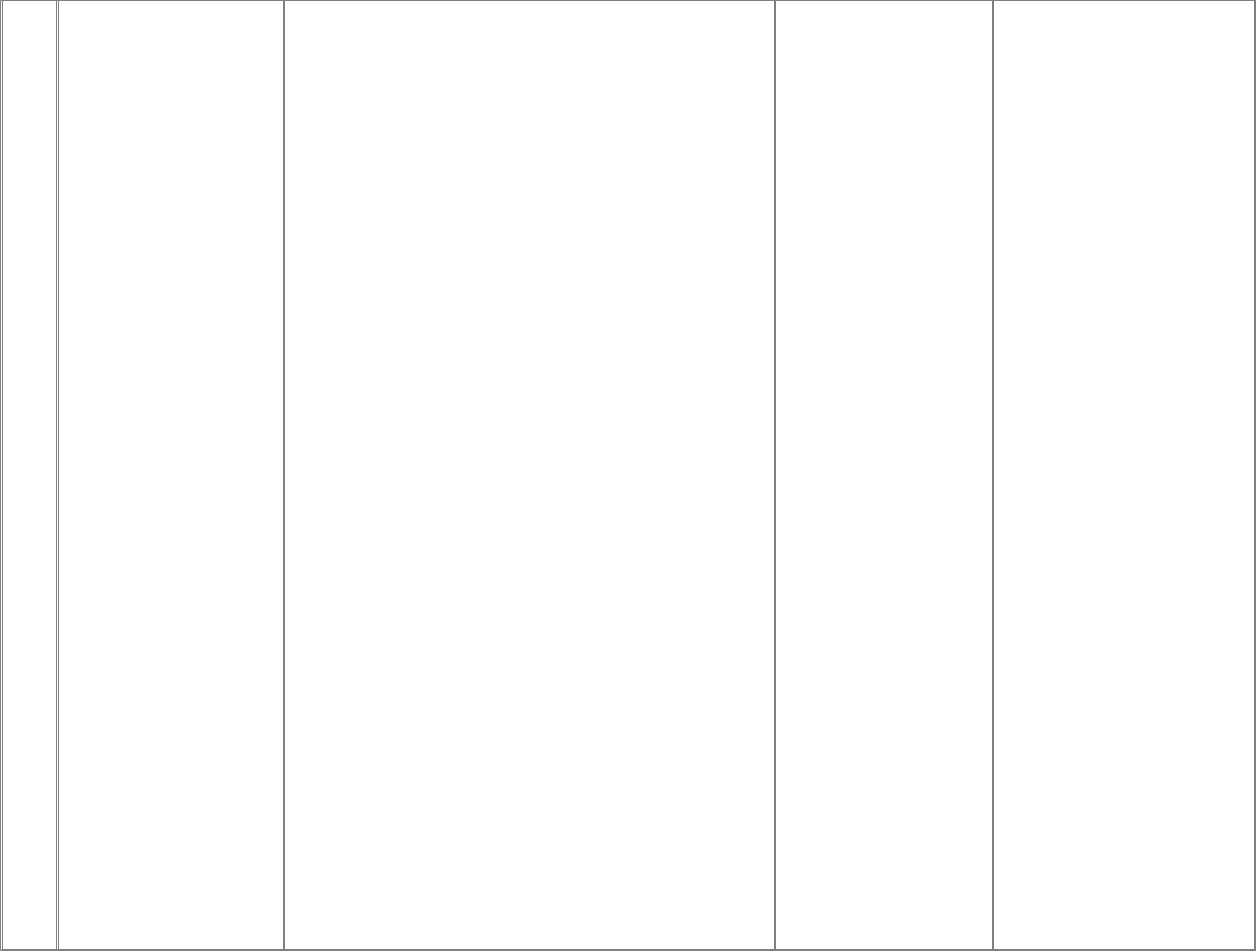
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** | **Комментарий** |  |
| 1 |  | Объявление строковой | 2 |  |  |
|  | переменной root\_object\_name |  |  |
| 2 |  | Ввод значения переменной | 3 |  |  |
|  | root\_object\_name |  |  |
|  |  |  |  |  |
| 3 |  | Вызов метода | 4 |  |  |

13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | define\_object\_name с |  |  |  |
|  |  | аргументом root\_object\_name |  |  |  |
|  |  | Вызов метода |  |  |  |
| 4 |  | define\_object\_state с | 5 |  |  |
|  |  | аргументом 1 |  |  |  |
| 5 |  | Объявление указателей child | 6 |  |  |
|  |  | и parent\_object |  |  |  |
|  |  | Объявление строковых |  |  |  |
| 6 |  | переменных | 7 |  |  |
|  | parent\_object\_name и |  |  |
|  |  |  |  |  |
|  |  | object\_name |  |  |  |
|  |  | Объявление целочисленных |  |  |  |
| 7 |  | переменных object\_class и | 8 |  |  |
|  |  | object\_state |  |  |  |
| 8 |  | Ввод значения переменной | 9 |  |  |
|  | parent\_object\_name |  |  |
|  |  |  |  |  |
|  | parent\_object |  | ∅ |  |  |
|  | \_name равен |  |  |  |
| 9 | "endtree" |  |  |  |  |
|  | Ввод значений переменных |  |  |  |
|  |  |  |  |  |
|  |  | object\_name, object\_class и | 10 |  |  |
|  |  | object\_state |  |  |  |
|  |  | Направление указателя |  |  |  |
|  |  | parent\_object на объект, |  |  |  |
|  |  | возвращенный в результате |  |  |  |
| 10 |  | вызова метода | 11 |  |  |
|  |  | get\_object\_by\_path с |  |  |  |
|  |  | аргументом |  |  |  |
|  |  | parent\_object\_name |  |  |  |
| 11 |  | Направление child на |  |  |  |
|  | object\_class | динамически созданный |  |  |  |
|  | объект класса Class2 с | 12 |  |  |
|  | равен 2 |  |  |
|  | параметрами вида |  |  |  |
|  |  |  |  |  |
|  |  | parent\_object и object\_name |  |  |  |
|  | object\_class | Направление child на | 12 |  |  |
|  | равен 3 | динамически созданный |  |  |  |
|  |  | объект класса Class3 с |  |  |  |
|  |  | параметрами вида |  |  |  |

14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | parent\_object и object\_name |  |  |
|  |  | Направление child на |  |  |
|  | object\_class | динамически созданный |  |  |
|  | объект класса Class4 с | 12 |  |
|  | равен 4 |  |
|  | параметрами вида |  |  |
|  |  |  |  |
|  |  | parent\_object и object\_name |  |  |
|  |  | Направление child на |  |  |
|  | object\_class | динамически созданный |  |  |
|  | объект класса Class5 с | 12 |  |
|  | равен 5 |  |
|  | параметрами вида |  |  |
|  |  |  |  |
|  |  | parent\_object и object\_name |  |  |
|  |  | Направление child на |  |  |
|  | object\_class | динамически созданный |  |  |
|  | объект класса Class6 с | 12 |  |
|  | равен 6 |  |
|  | параметрами вида |  |  |
|  |  |  |  |
|  |  | parent\_object и object\_name |  |  |
|  |  |  | 8 |  |
|  |  | Вызов метода |  |  |
| 12 |  | define\_object\_state объекта, на | 8 |  |
|  |  | который направлен child, с |  |  |
|  |  | аргументом object\_state |  |  |



Класс объекта: BaseApplication

Модификатор доступа: public

Метод: exec\_app

Функционал: запускает приложение

Параметры: нет

Возвращаемое значение: int, код возврата

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода exec\_app класса BaseApplication



 **№**  **Предикат**

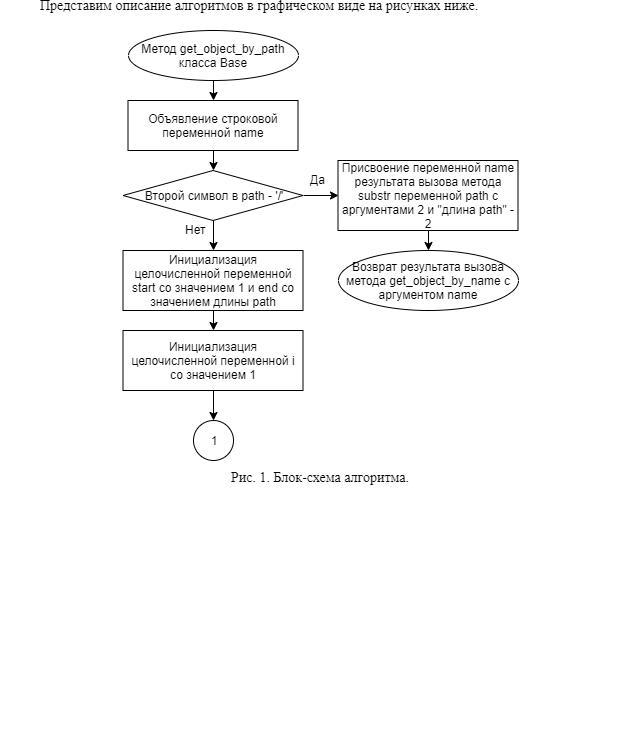
|  |  |  |
| --- | --- | --- |
| **Действия** | **№ перехода** | **Комментарий** |

15

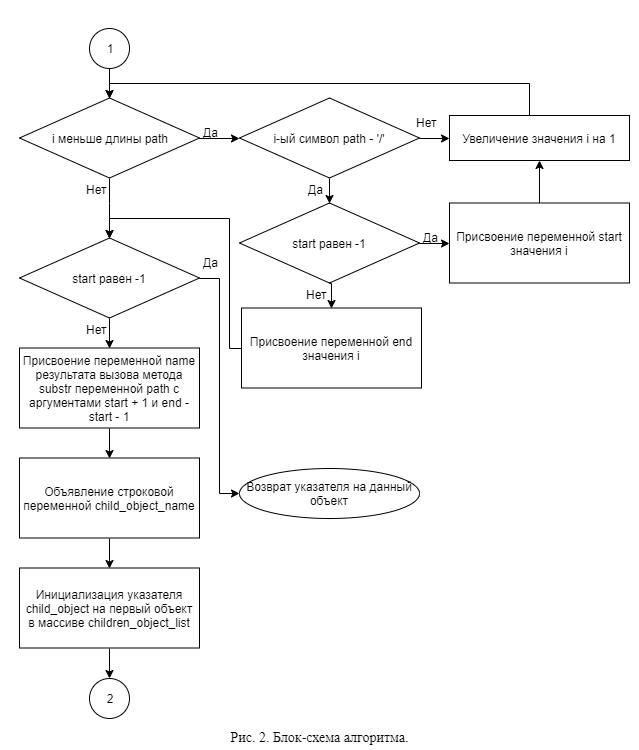
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 |  | Вывод "Object tree" и | 2 |  |  |
|  | переход на новую строку |  |  |
|  |  |  |  |  |
| 2 |  | Вызов метода | 3 |  |  |
|  | print\_hierarchy\_tree |  |  |
|  |  |  |  |  |
| 3 |  | Объявление строковой | 4 |  |  |
|  | переменной coordinate |  |  |
|  |  |  |  |  |
| 4 |  | Ввод значения переменной | 5 |  |  |
|  |  | coordinate |  |  |  |
| 5 |  | Объявление указателя | 6 |  |  |
|  | coordinate\_object |  |  |
|  |  |  |  |  |
|  |  | Направление указателя |  |  |  |
|  |  | coordinate\_object на объект, |  |  |  |
|  | coordinate не | возвращенный в результате | 7 |  |  |
| 6 | равен "//" | вызова метода |  |  |  |
|  |  | get\_object\_by\_path с |  |  |  |
|  |  | аргументом coordinate |  |  |  |
|  |  |  | 10 |  |  |
|  |  | Переход на новую строку и |  |  |  |
| 7 |  | вывод значения переменной | 8 |  |  |
|  |  | coordinate |  |  |  |
|  |  | Вывод " Object name: " и |  |  |  |
|  | coordinate\_obje | результата вызова метода |  |  |  |
| 8 | ct не нулевой | get\_object\_name объекта, на | 9 |  |  |
| указатель | который указывает |  |  |  |
|  |  | coordinate\_object |  |  |  |
|  |  | Вывод " Object not found" | 9 |  |  |
| 9 |  | Ввод значения переменной | 6 |  |  |
|  |  | coordinate |  |  |  |
| 10 |  | Возврат 0 | ∅ |  |  |

16

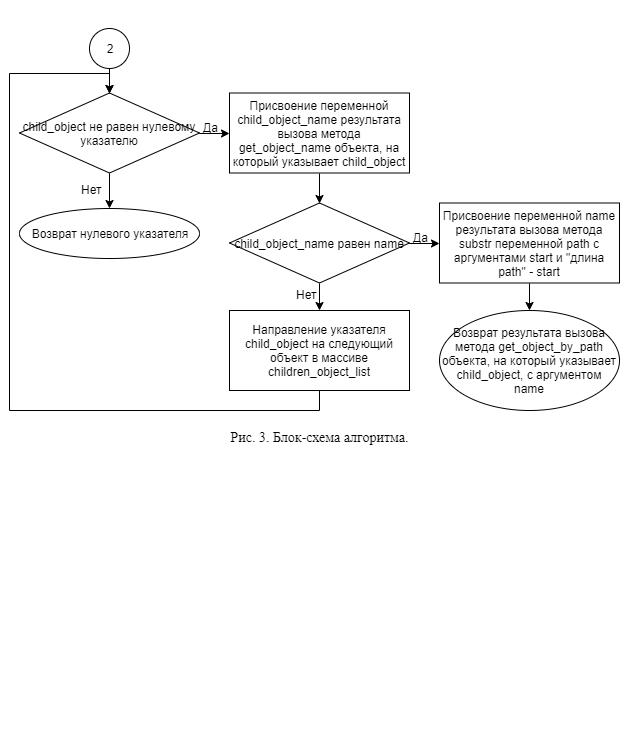
**Блок-схема алгоритма**



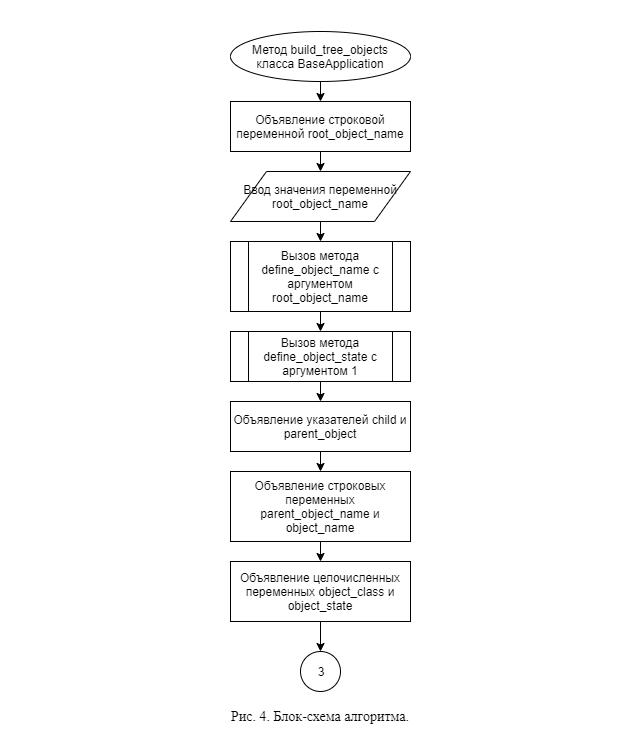
17



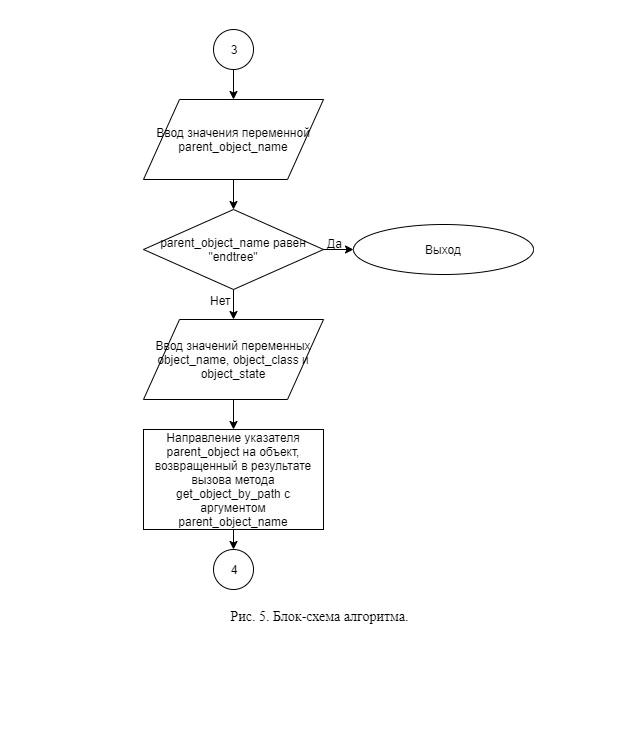
18



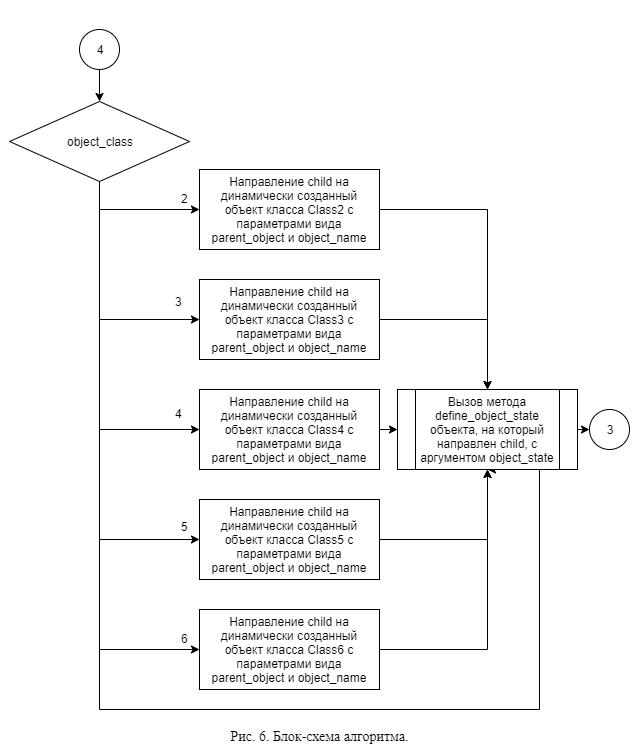
19



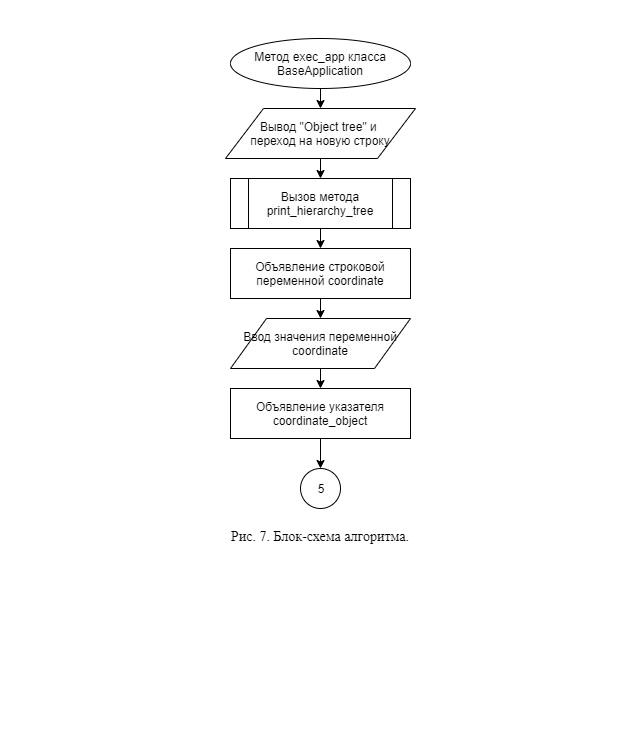
20



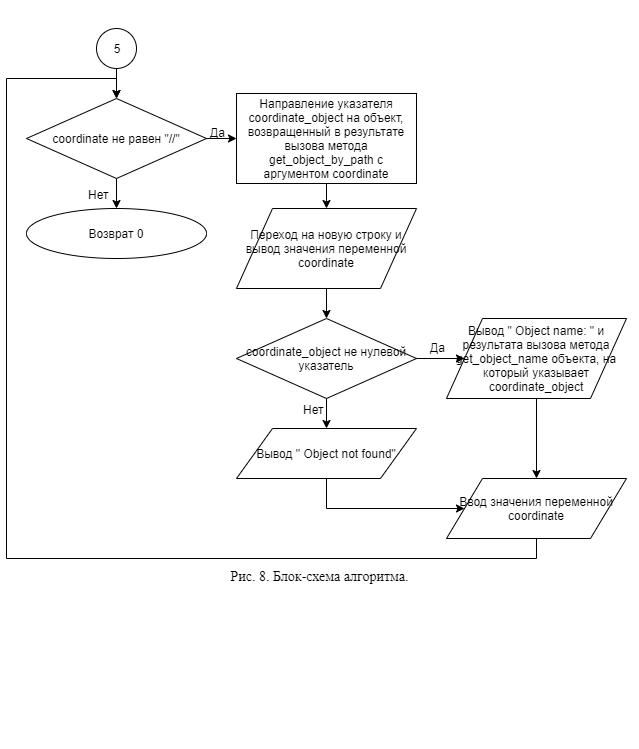
21



22



23



24

**Код программы**

Программная реализация алгоритмов для решения задачи представлена ниже.

**Файл BaseApplication.cpp**

#include "BaseApplication.h"

#include "Class2.h"

#include "Class3.h"

#include "Class4.h"

#include "Class5.h"

#include "Class6.h"

#include <iostream>

#include <string>

void BaseApplication::build\_tree\_objects() {

std::string root\_object\_name;

std::cin >> root\_object\_name;

define\_object\_name(root\_object\_name);

define\_object\_state(1);

Base\* child;

Base\* parent\_object;

std::string parent\_object\_name, object\_name; int object\_class, object\_state;

while (true) {

std::cin >> parent\_object\_name;

if (parent\_object\_name == "endtree") {

break;

}

std::cin >> object\_name >> object\_class >> object\_state; parent\_object = get\_object\_by\_path(parent\_object\_name); switch (object\_class) {

case 2:

child = new Class2(parent\_object, object\_name); child->define\_object\_state(object\_state); break;

case 3:

child = new Class3(parent\_object, object\_name); child->define\_object\_state(object\_state); break;

case 4:

child = new Class4(parent\_object, object\_name); child->define\_object\_state(object\_state); break;

case 5:

child = new Class5(parent\_object, object\_name); child->define\_object\_state(object\_state); break;

case 6:

child = new Class6(parent\_object, object\_name);

25

child->define\_object\_state(object\_state); break;

}

}

}

int BaseApplication::exec\_app() {

std::cout << "Object tree\n";

print\_hierarchy\_tree();

std::string coordinate;

std::cin >> coordinate;

Base\* coordinate\_object;

while (coordinate != "//") {

coordinate\_object = get\_object\_by\_path(coordinate); std::cout << "\n" << coordinate;

if (coordinate\_object != nullptr) {

std::cout << " Object name: " << coordinate\_object-

>get\_object\_name();

}

else {

std::cout << " Object not found";

}

std::cin >> coordinate;

}

return 0;

}

**Файл BaseApplication.h**

#ifndef BASE\_APPLICATION\_H\_

#define BASE\_APPLICATION\_H\_

#include "Base.h"

class BaseApplication: public Base {

public:

using Base::Base;

void build\_tree\_objects();

int exec\_app();

};

#endif // BASE\_APPLICATION\_H\_

**Файл Base.cpp**

26

#include "Base.h"

#include <algorithm>

#include <iostream>

#include <string>

#include <vector>

Base::Base(Base\* parent, std::string name)

* object\_parent(parent), object\_name(name) { if (parent != nullptr) {

parent->children\_object\_list.push\_back(this);

}

}

Base::~Base() {

for (auto child\_object : children\_object\_list) { delete child\_object;

}

}

void Base::define\_object\_name(std::string name) { object\_name = name;

}

std::string Base::get\_object\_name() {

return object\_name;

}

void Base::print\_object\_tree() {

if (object\_parent == nullptr) {

std::cout << object\_name;

}

if (!children\_object\_list.empty()) { std::cout << "\n" << object\_name;

for (auto child\_object : children\_object\_list) { std::cout << " " << child\_object->object\_name;

}

for (auto child\_object : children\_object\_list) { child\_object->print\_object\_tree();

}

}

}

void Base::redefine\_object\_parent(Base\* new\_parent) {

remove(object\_parent->children\_object\_list.begin(), object\_parent-

>children\_object\_list.end(), this);

object\_parent = new\_parent;

new\_parent->children\_object\_list.push\_back(this);

}

Base\* Base::get\_object\_parent() {

return object\_parent;

}

int Base::get\_object\_state() {

return object\_state;

}

void Base::define\_object\_state(int state) { object\_state = state;

27

}

void Base::print\_state\_tree() {

std::string not\_ready = "";

if (object\_state <= 0) {

not\_ready = "not ";

}

std::cout << "The object " << object\_name << " is " << not\_ready <<

"ready";

if (!children\_object\_list.empty()) {

std::cout << "\n";

}

for (auto child\_object : children\_object\_list) { child\_object->print\_state\_tree();

if (child\_object != children\_object\_list.back()) { std::cout << "\n";

}

}

}

Base\* Base::get\_object\_by\_name(std::string name) { if (object\_name == name) {

return this;

}

Base\* search\_result;

for (auto child\_object : children\_object\_list) {

search\_result = child\_object->get\_object\_by\_name(name);

if (search\_result != nullptr) {

return search\_result;

}

}

return nullptr;

}

void Base::print\_hierarchy\_tree(int level) {

std::cout << std::string(level \* 4, ' ') << get\_object\_name(); if (!children\_object\_list.empty()) {

std::cout << "\n";

}

for (auto child\_object : children\_object\_list) { child\_object->print\_hierarchy\_tree(level + 1);

if (child\_object != children\_object\_list.back()) { std::cout << "\n";

}

}

}

Base\* Base::get\_object\_by\_path(std::string path) { std::string name;

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

name = path.substr(2, path.length() - 2); return get\_object\_by\_name(name);

}

28

int start = -1, end = path.length();

for (int i = 1; i < path.length(); i++) { if (path[i] == '/') {

if (start == -1) {

start = i;

}

else {

end = i;

break;

}

}

}

if (start == -1) {

return this;

}

name = path.substr(start + 1, end - start - 1);

std::string child\_object\_name;

for (auto child\_object : children\_object\_list) {

child\_object\_name = child\_object->get\_object\_name();

if (child\_object\_name == name) {

name = path.substr(start, path.length() - start); return child\_object->get\_object\_by\_path(name);

}

}

return nullptr;

}

**Файл Base.h**

#ifndef BASE\_H\_

#define BASE\_H\_

#include <string>

#include <vector>

class Base {

public:

Base(Base\* parent, std::string name="Object\_root"); virtual ~Base();

void define\_object\_name(std::string name); std::string get\_object\_name();

void print\_object\_tree();

void redefine\_object\_parent(Base\* new\_parent); Base\* get\_object\_parent();

int get\_object\_state();

void define\_object\_state(int state);

void print\_state\_tree();

Base\* get\_object\_by\_name(std::string name); void print\_hierarchy\_tree(int level=0); Base\* get\_object\_by\_path(std::string path);

protected:

std::string object\_name;

Base\* object\_parent;

std::vector<Base\*> children\_object\_list;

29

int object\_state;

};

#endif // BASE\_H\_

**Файл Class2.h**

#ifndef CLASS2\_H\_

#define CLASS2\_H\_

#include "Base.h"

class Class2: public Base {

public:

using Base::Base;

};

#endif // CLASS2\_H\_

**Файл Class3.h**

#ifndef CLASS3\_H\_

#define CLASS3\_H\_

#include "Base.h"

class Class3: public Base {

public:

using Base::Base;

};

#endif // CLASS3\_H\_

**Файл Class4.h**

#ifndef CLASS4\_H\_

#define CLASS4\_H\_

#include "Base.h"

class Class4: public Base {

public:

using Base::Base;

};

30

#endif // CLASS4\_H\_

**Файл Class5.h**

#ifndef CLASS5\_H\_

#define CLASS5\_H\_

#include "Base.h"

class Class5: public Base {

public:

using Base::Base;

};

#endif // CLASS5\_H\_

**Файл Class6.h**

#ifndef CLASS6\_H\_

#define CLASS6\_H\_

#include "Base.h"

class Class6: public Base {

public:

using Base::Base;

};

#endif // CLASS6\_H\_

**Файл main.cpp**

#include "BaseApplication.h"

int main()

{

* program here BaseApplication base(nullptr); base.build\_tree\_objects(); return base.exec\_app();

}

31

**Тестирование**

Результат тестирования программы представлен в следующей таблице.

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемые выходные | Фактические выходные |
|  | данные | данные |
|  |  |  |
| root endtree //root // | Object tree root //root | Object tree root //root |
|  | Object name: root | Object name: root |
|  |  |  |
| root /root object\_1 3 1 | Object tree root object\_1 | Object tree root object\_1 |
| /root object\_2 2 1 | object\_7 object\_2 object\_4 | object\_7 object\_2 |
| /root/object\_2 object\_4 3 - | object\_7 object\_5 object\_3 | object\_4 object\_7 |
| 1 /root/object\_2 object\_5 4 | object\_3 //object\_2 Object | object\_5 object\_3 |
| 1 /root object\_3 3 1 | name: object\_2 //root | object\_3 //object\_2 Object |
| /root/object\_2 object\_3 6 1 | Object name: root | name: object\_2 //root |
| /root/object\_1 object\_7 5 1 | //object\_69 Object not | Object name: root |
| /root/object\_2/object\_4 | found | //object\_69 Object not |
| object\_7 3 -1 endtree |  | found |
| //object\_2 //root |  |  |
| //object\_69 // |  |  |
|  |  |  |

32

**ЗАКЛЮЧЕНИЕ**

Курс ООП познакомил меня с данной парадигмой программирования и научил многим навыкам. В начале я разрабатывал базовые классы для объектов и определял общий функционал для используемых в рамках приложения объектов. Позднее я научился разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева и строить дерево иерархии объектов. К концу курса я использовал алгоритмы обработки структур данных в виде дерева, переключал состояния объектов и определял их готовность к работе, а также выводил на печать дерево иерархии объектов и искал указатель на объект по координате по дереву иерархии объектов или по имени, при уникальности наименований объектов.

Выполнение данной курсовой работы позволило мне закрепить навыки выше и использовать их свободно при программировании.

33

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

**(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. —
3. с.
4. 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).
5. 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).
6. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».
7. 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).

34