

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Департамент программной инженерии и искусственного интеллекта**

ОРЛОВ ГЕОРГИЙ АЛЕКСАНДРОВИЧ

СЕРВИС ЗНАКОМСТВ «ВМЕСТЕ»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Фундаментальные структуры данных и алгоритмы»

по образовательной программе подготовки бакалавров по направлению

09.03.04 - Программная инженерия

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |  | Студент гр. Б9120-09.03.04прогин | | |
|  |  |  | | Г.А. Орлов |
|  | | | | | | |  |  | (подпись) | |  |
|  | | | | | | | | | | | |
| Защищен с оценкой | | | | | | |  |  | Руководитель | | |
|  | | | | | | |  |  | ст. преподаватель | | |
|  | | | |  |  | |  |  | ученая степень, должность |  | О.А. Крестникова |
| (подпись) | | | |  | (И.О. Фамилия) | |  |  | (подпись) |  | (И.О. Фамилия) |
| « |  | » |  | | | 2022 г. |  |  |  | | |
|  | | | | | | | | | | | |

г. Владивосток

2022

# Оглавление

[Оглавление 2](#_Toc116598914)

[Введение 3](#_Toc116598915)

[1 Анализ предметной области 4](#_Toc116598916)

[1.1 Объекты предметной области 4](#_Toc116598917)

[1.2 Законы ПО 6](#_Toc116598918)

[1.3 Постановка задачи «Формирование отчета «Сервис знакомств»» 6](#_Toc116598919)

[2 Теоретическая часть 7](#_Toc116598920)

[2.1 Хеш-таблица 7](#_Toc116598921)

[2.1.1 Хеш-функция 7](#_Toc116598922)

[2.1.2 Разрешение коллизий методом открытой адресации (квадратичный) 8](#_Toc116598923)

[2.2 АВЛ дерево 9](#_Toc116598924)

[2.2.1 Односвязный список, для хранения дублей 10](#_Toc116598925)

[3 Требования к информационной системе 11](#_Toc116598926)

[3.1 Требования к данным 11](#_Toc116598927)

[3.1.1 Требования к входным данным 11](#_Toc116598928)

[3.1.2 Требования к выходным данным 12](#_Toc116598929)

[3.2 Функциональные требования 13](#_Toc116598930)

[4 Реализация 14](#_Toc116598931)

[4.1 Спецификация структур данных 14](#_Toc116598932)

[4.2 Описание среды разработки 21](#_Toc116598933)

[4.3 Руководство пользователя 21](#_Toc116598934)

[4.4 Тестирование 27](#_Toc116598935)

[Заключение 28](#_Toc116598936)

[Список литературы 29](#_Toc116598937)

# Введение

Студенты, живущие на кампусе, нередко испытывают проблемы, при поиске партнёра. Для решения этой проблемы был разработан проект «Сервис знакомств Вместе»

Целью курсового проекта является: разработка информационной системы в виде социальной сети, с представлением пользователей и их публикаций.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области «Сервис знакомств».

2. Изучить теоретические основы методов построения справочников.

3. Определить требования к информационной системе.

4. Реализовать и провести тестирование.

# 1 Анализ предметной области

Требуется разработать информационную систему для предметной области (ПО) «Пользователи», и ПО «Публикации пользователей».

Система должна решать следующие задачи:

1. хранить, позволять просматривать, добавлять и удалять информацию о пользователе;
2. позволять искать информацию о пользователях, по их адресу;
3. использовать пакетную загрузку данных о пользователях, из текстового файла
4. хранить, позволять просматривать, добавлять и удалять публикации пользователей;
5. позволять искать информацию о публикациях пользователей, по их дате;
6. использовать пакетную загрузку данных о публикациях пользователей, из текстового файла

1.1 Объекты предметной области

Объект– информация о нём содержит: ФИО, логин, пол и адрес пользователя

Объект **Публикация пользователя** – информация о нём содержит: дату, текст, изображение, и логин пользователя

Исходя из задач, которые должна решать информационная система, выделены объекты ПО, представленные в Таблице 1.

Таблица 1 – Объекты ПО

|  |  |
| --- | --- |
| Тип и название объекта | Описание объекта |
| Справочник «Пользователи» | Хранит информацию по каждому пользователю, а именно хранит поля: ФИО, логин, пол и адрес |
| Справочник «Публикации пользователей» | Хранит информацию о каждой публикации каждого пользователя, а именно хранит поля: дата, текст, изображение и логин |

**ФИО** – аббревиатура, расшифровывается как: фамилия, имя, отчество. Фамилия, имя и отчество являются словами. Слова разделены одним пробелом. Максимальная длинна – 30 символов.

**Логин** – это одно или несколько слов, из букв английского алфавита, и цифр. Для каждого пользователя логин является уникальным, т.е. не может быть двух пользователей, с одинаковыми логинами. Максимальная длинна – 20 символов.

**Пол –** это последовательность, содержащая от 1 до 3х символов, и характеризующая половую принадлежность, к той или иной половой группе, данного пользователя. Максимальная длинна – 3 символа.

**Адрес –** это набор слов и символов, указывающий на местоположение данного пользователя. Максимальная длинна – 25 символов.

**Дата –** это структура, содержащая год, месяц, и день. Все поля числовые, корректность даты не проверяется.

**Текст –** это текстовое поле, содержащее текстовые символы. Максимальная длинна – 20 символов.

**Изображение** – это текстовое поле, содержащее имя изображения. Максимальная длинна – 12 символов.

Пример справочника «Пользователи» представлен в таблице 2.

Пример справочника «Публикации пользователей» представлен в таблице 3.

Таблица 2 – Справочник «Пользователи»

|  |  |  |  |
| --- | --- | --- | --- |
| **ФИО** | **Логин** | **Пол** | **Адрес** |
| 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 1 |
| 1 | 3 | 1 | 1 |
| 1 | 6 | 1 | 1 |
| 1 | 7 | 1 | 1 |
| 1 | 8 | 1 | 2 |
| 1 | 9 | 1 | 3 |
| 1 | 10 | 1 | 3 |
| 1 | 12 | 1 | 3 |
| 1 | 14 | 1 | 4 |
| 1 | 21 | 1 | 5 |

Таблица 3 – Справочник «Публикации пользователей»

|  |  |  |  |
| --- | --- | --- | --- |
| **Дата** | **Текст** | **Изображение** | **Логин** |
| 11.7.2019 | Text\_1 | Image\_1 | **1** |
| 26.1.2014 | Text\_2 | Image\_2 | **2** |
| 6.11.2018 | Text\_3 | Image\_3 | **3** |
| 11.7.2009 | Text\_6 | Image\_6 | **6** |
| 26.1.2004 | Text\_7 | Image\_7 | **7** |
| 6.11.2008 | Text\_8 | Image\_8 | **8** |
| 16.12.2007 | Text\_9 | Image\_9 | **9** |
| 9.2.2006 | Text\_10 | Image\_10 | **10** |

1.2 Законы ПО

1. У пользователя может быть сколько угодно много публикаций;
2. Не может быть публикации у несуществующего пользователя;
3. Каждый логин пользователя уникален;

1.3 Постановка задачи «Формирование отчета «Сервис знакомств»»

Входные данные: Справочник «Пользователи», Справочник «Публикации пользователей», Поле адреса, при поиске в справочнике Пользователей, Поле даты, при поиске в справочнике Публикаций пользователей

Выходные данные: Отчет «Пользователи», Отчет «Публикации пользователей».

Связь:

Если выполнены условия поиска:

Пользователь.Адрес = Адрес

Публикации пользователей.Дата = Дата

Отчёт Пользователи.Адрес = Адрес

Отчёт Публикации пользователей.Дата = Дата

# 2 Теоретическая часть

В своём проекте я использую 3 структуры, для хранения и быстрого доступа к данным.

Хеш-таблица – это структура данных, которая позволяет, не задействуя поиск, моментально обращаться к нужному элементу. Однако, она использует ресурсы памяти. В моём проекте использование хеш-таблицы логично, и обоснованно. [2].

Поиск в АВЛ дереве – самый быстрый из всех бинарных деревьев. А алгоритм вставки также самый быстрая, однако занимает больше памяти. Моя основная задача – это поиск, и поэтому я использую АВЛ дерево. [1].

Для хранения дублей я использую односвязный динамический список. Эта структура позволяет легко организовать хранение данных, если их количество изначально неизвестно. [3].

2.1 Хеш-таблица

Хэш-таблица представляет собой структуру данных, которая реализует абстрактную структуру, которая может сопоставлять ключи со значениями. Хеш-таблица использует хеш-функцию для вычисления индекса, в массив ячеек, из которых можно найти желаемое значение. Во время поиска ключ хэшируется, и полученный хэш указывает, где хранится соответствующее значение. [2].

2.1.1 Хеш-функция

Хеш-функция – это функция, используемая для вычисления номера ячейки в хеш-таблице для заданного ключа. Цель хеш-функции состоит в том, чтобы уменьшить диапазон индексов массив

Хеш-функция, используемая в моей реализации Хеш-Таблицы – это середина квадрата. Входное значение преобразовывается в число, возводится в квадрат, затем берутся две, или одна цифра из середины полученного числа, и приводятся к размеру Хеш-Таблицы. Так находится значение ключа. [2].

Пример:

На вход подаётся дата, в виде строки. Например: 26.1.2004

Далее вычисляется сумма кодов всех символов.

Потом она возводится в квадрат, и берётся 1 или 2 цифры из середины этого числа (это зависит от количества цифр в числе, если оно чётное, то берётся 2 цифры, если нет – то одна)

Затем полученное число приводится к размеру Хеш-таблицы. Это и будет первичная хеш-функция

Если наша Хеш-Таблица имеет размер 32, то:

Hash(26.1.2004) => 443^2 = 196 249 => 62%16 = 2

2.1.2 Разрешение коллизий методом открытой адресации (квадратичный)

Работает по принципу того, что расстояние между занятыми ячейками становится больше одного, из-за этого уменьшается вероятность вырождения ХТ в цепочку, при перехеше, когда каждый предыдущий элемент ссылается вторичным значением ХФ на следующей.

Метод разрешения коллизий используя квадратичную открытую адресацию, устраняет этот недостаток. И благодаря ему вторичных хеш ссылается на вычислимое, но случайное место в ХТ. [2].

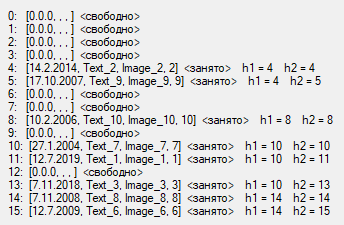


Рисунок 1 – Окно Debug Хеш-Таблицы

2.2 АВЛ дерево

Сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины. [1].

Используется 4 типа вращений:

Малое левое, большое левое, малое правое, большое правое.

Операции работы с АВЛ деревом: Добавление, удаление, очистка, поиск по ключу.

При удалении замена на минимальный справа

При добавлении и удалении любой вершины запускается балансировка. Она проходит по каждой вершине, баланс которой мог быть изменён в результате операции. Если у вершины баланс будет больше 1, или меньше -1, то используется один из четырёх поворотов, для выравнивания баланса.

Пример дерева на тестовых данных:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Окно Debug АВЛ-Дерева

2.2.1 Односвязный список, для хранения дублей

Для хранения дублей данных, по ключу, в АВЛ дереве я использую структуру односвязного динамического списка.

Односвязный динамический список – это структура, каждый элемент которой имеет 2 поля: указатель на следующий элемент (next), и указатель на хранимые данные (data). Указатель на начало списка (head) хранится отдельно. [3].

На первый элемент всегда указывает указатель начала списка (head), и каждый элемент указывает на следующий (next). Если элемент последний в списке то он указывает на null. [3].

# 3 Требования к информационной системе

Здесь описаны основные требования к конечному информационному продукту. Все эти требования были выполнены.

3.1 Требования к данным

3.1.1 Требования к входным данным

Основываясь на анализе ПО, входными данными является:

* текстовый файл TextFile\_1.gog1, каждая строка файла содержит информацию об объекте пользователь, а именно ФИО, логин, пол и адрес, разделенный символом запятая;

Пример текстового файла:

Орлов Георгий Александрович, gogortey, М, ул. Державина 21,   
Кравцов Максим Павлович, maksim.krav, М, ул. Матрёшкина 43,   
Исаева Варвара Ярославовна, aveyasi, Ж, ул. Иваненко 19,   
Поляков Ярослав Кириллович, polyakov04, М, ул. Зайцев 44,   
Окулова Мирослава Максимовна, miroslava, Ж, ул. Вермеева 12,   
Леонова Ульяна Александровна, ulyana.leonova, Ж, ул. Сакософонова 4,   
Крючкова Елизавета Ильинична, ki.yelizaveta, Ж, ул. Вертолётная 11,   
Прохорова Милана Данииловна, milana, Ж, ул. Ложкова 7,   
Филимонов Дмитрий Александрович, fil.dmitry, М, ул. Центральная 29,   
Иванова Марина Кирилловна, marina.i, Ж, ул. Героев 14,

* текстовый файл TextFile\_3.gog2, каждая строка файла содержит информацию об объекте пост пользователя, а именно дату, текст, имя изображения, и логин пользователя, разделенный символом запятая;

Пример текстового файла:

11.7.2019, Text\_1, Image\_1, 1,

26.1.2014, Text\_2, Image\_2, 2,

6.11.2018, Text\_3, Image\_3, 3,

16.12.2017, Text\_4, Image\_4, 4,

9.2.2016, Text\_5, Image\_5, 5,

11.7.2009, Text\_6, Image\_6, 6,

26.1.2004, Text\_7, Image\_7, 7,

6.11.2008, Text\_8, Image\_8, 8,

16.12.2007, Text\_9, Image\_9, 9,

9.2.2006, Text\_10, Image\_10, 10,

3.1.2 Требования к выходным данным

* текстовый файл TextFile\_1.gog1, каждая строка файла содержит информацию об объекте пользователь, а именно ФИО, логин, пол и адрес, разделенный символом запятая;

Пример текстового файла:

Орлов Георгий Александрович, gogortey, М, ул. Державина 21,   
Кравцов Максим Павлович, maksim.krav, М, ул. Матрёшкина 43,   
Исаева Варвара Ярославовна, aveyasi, Ж, ул. Иваненко 19,   
Поляков Ярослав Кириллович, polyakov04, М, ул. Зайцев 44,   
Окулова Мирослава Максимовна, miroslava, Ж, ул. Вермеева 12,   
Леонова Ульяна Александровна, ulyana.leonova, Ж, ул. Сакософонова 4,   
Крючкова Елизавета Ильинична, ki.yelizaveta, Ж, ул. Вертолётная 11,   
Прохорова Милана Данииловна, milana, Ж, ул. Ложкова 7,   
Филимонов Дмитрий Александрович, fil.dmitry, М, ул. Центральная 29,   
Иванова Марина Кирилловна, marina.i, Ж, ул. Героев 14,

* текстовый файл TextFile\_3.gog2, каждая строка файла содержит информацию об объекте пост пользователя, а именно дату, текст, имя изображения, и логин пользователя, разделенный символом запятая;

Пример текстового файла:

11.7.2019, Text\_1, Image\_1, 1,

26.1.2014, Text\_2, Image\_2, 2,

6.11.2018, Text\_3, Image\_3, 3,

16.12.2017, Text\_4, Image\_4, 4,

9.2.2016, Text\_5, Image\_5, 5,

11.7.2009, Text\_6, Image\_6, 6,

26.1.2004, Text\_7, Image\_7, 7,

6.11.2008, Text\_8, Image\_8, 8,

16.12.2007, Text\_9, Image\_9, 9,

9.2.2006, Text\_10, Image\_10, 10,

3.2 Функциональные требования

Информационная система, должна позволять:

* считывать данные справочников из текстовых файлов (см п 3.1.2);
* добавлять информацию в справочник «Пользователи» по значениям: ФИО, логин, пол, адрес;
* проверять добавляемые значения в справочник «Пользователи» на корректность, а именно: для каждого текстового поля длинна вводимой строки больше 1 символа, и меньше 40;
* добавлять информацию в справочник «Публикации пользователей» по значениям: дата, текст, картинка, логин;
* проверять добавляемые значения в справочник «Публикации пользователей» на корректность, а именно: для каждого текстового поля длинна вводимой строки – больше 1 символа, и меньше 40;
* проверять на уникальность логин, введённый пользователем, при добавлении в справочник «Пользователи»;
* проверять на уникальность дату и логин, введённый пользователем, при добавлении в справочник «Публикации пользователей»;
* соблюдать целостность между справочниками «Пользователи» и «Публикации пользователей» так, что при удалении пользователя, удаляются все его публикации;
* находить и удалять информацию из справочника «Пользователи», по значению поля адрес;
* находить и удалять информацию из справочника «Пользователи», по значениям: дата, адрес;
* сохранять все внесённые изменения в справочники «Пользователи» и «Публикации пользователей» в файлы (см. п. 3.1.2);
* создавать отчет «Общий» по заданному логину и дате;

# 4 Реализация

4.1 Спецификация структур данных

Класс **AVL** - класс, описывающий АВЛ-дерево.

Поля:

* Класс **Node** – класс, описывающий узел дерева;
* Поля:
  + List<T2> chain – кольцевой двусвязный список;
  + Int height – переменная, отвечающая за высоту поддерева;
  + T1 key – переменная, хранит ключ;
  + Node \*Left – указатель на левый потомок узла дерева;
  + Node \*Right - указатель на правый потомок узла дерева

Методы:

– void insert(T1 val\_1, T2 val\_2) – вставка элемента в дерево;

Входные данные: дерево, элемент, который хотим вставить, состоящий из двух полей: ключ, значение;

Формальные параметры: переменная T1 val\_1 – ключ, по которому вставляется элемент, переменная T2 val\_2 – значение, которое вставляется в дерево;

Выходные данные: дерево с добавленным элементом.

– void remove(T1 val\_1, T2 val\_2) – удаление элемента из дерева;

Входные данные: дерево, элемент, который хотим удалить, состоящий из двух полей: ключ, значение;

Формальные параметры: переменная T1 val\_1 – ключ, по которому удаляется элемент, переменная T2 val\_2 – значение, которое удаляется из дерева;

Выходные данные: дерево с удаленным элементом.

– Node\* search(T1 val\_1) – возвращает указатель на узел по заданному ключу;

Входные данные: дерево, ключ поиска;

Формальные параметры: переменная T1 val\_1 – ключ, по которому ищется узел;

Выходные данные: указатель на узел, имеющий заданный ключ. Если узла нет, то null.

– void inorder() – обход дерева;

Входные данные: дерево;

Формальные параметры: отсутствуют ;

Выходные данные: выведенное в консоль поэлементно дерево.

– void print() – вывод дерева в отладчик;

Входные данные: дерево;

Формальные параметры: отсутствуют ;

Выходные данные: выведенное в поле отладки дерево.

– vector<Node\*> get\_nodes\_higher(T1 val\_1) – возвращает вектор указателей на узлы, которые по ключу больше заданного ключа;

Входные данные: дерево, ключ поиска;

Формальные параметры: переменная T1 val\_1 – ключ, по которому ищутся узлы;

Выходные данные: вектор указателей на узлы, ключ которых больше заданного.

– vector<T2> to\_vector\_second\_value() – возвращает вектор всех значений узлов дерева;

Входные данные: дерево;

Формальные параметры: отсутствуют;

Выходные данные: вектор всех значений узлов дерева.

– Node\* rightRotation(Node\* head) – выполняет правый поворот в заданном узле дерева;

Входные данные: узел, в котором происходит поворот;

Формальные параметры: переменная Node\* head – узел, в котором происходит балансировка;

Выходные данные: сбалансированный узел.

– Node\* leftRotation(Node\* head) – выполняет левый поворот в заданном узле дерева;

Входные данные: узел, в котором происходит поворот;

Формальные параметры: переменная Node\* head – узел, в котором происходит балансировка;

Выходные данные: сбалансированный узел.

– void inorderUtil(Node\* head) – вспомогательный метод для метода void inorder();

Входные данные: дерево, указатель на узел, в котором на данный момент рекурсии происходит обход;

Формальные параметры: указатель Node\* head – указатель на узел, в котором на данный момент рекурсии происходит обход;

Выходные данные: выведенное в консоль поэлементно дерево.

– Node\* insertUtil(Node\* head, T1 val\_1, T2 val\_2) – вспомогательный метод для метода void insert(T1 val\_1, T2 val\_2);

Входные данные: дерево, ключ вставки, элемент вставки, указатель на узел, в котором на данный момент рекурсии находится метод;

Формальные параметры: указатель Node\* head – указатель на узел, в котором на данный момент рекурсии находится метод, переменная T1 val\_1 – ключ вставки, переменная T2 val\_2 – элемент вставки;

Выходные данные: дерево со вставленным элементом.

– Node\* removeUtil(Node\* head, T1 val\_1, T2 val\_2) – вспомогательный метод для метода void remove(T1 val\_1, T2 val\_2);

Входные данные: дерево, ключ удаления, элемент удаления, указатель на узел, в котором на данный момент рекурсии находится метод;

Формальные параметры: указатель Node\* head – указатель на узел, в котором на данный момент рекурсии находится метод, переменная T1 val\_1 – ключ удаления, переменная T2 val\_2 – элемент удаления;

Выходные данные: дерево с удаленным элементом.

– Node\* searchUtil(Node\* head, T1 val\_1) – вспомогательный метод для метода Node\* search(T1 val\_1);

Входные данные: указатель на узел, в котором на данный момент рекурсии находится метод, ключ поиска;

Формальные параметры: указатель Node\* head – указатель на узел, в котором на данный момент рекурсии находится метод, переменная T1 val\_1 – ключ поиска;

Выходные данные: найденный узел.

– void \_print(Node\* curr, int h) – вспомогательный метод для метода void print();

Входные данные: указатель на узел, в котором на данный момент рекурсии находится метод, целочисленное число – количество пробелов для данной итерации рекурсии;

Формальные параметры: указатель Node\* curr – указатель на узел, в котором на данный момент рекурсии находится метод, переменная int h – количество пробелов для данной итерации рекурсии;

Выходные данные: выведенное в поле отладки дерево.

– void \_get\_nodes\_higher(T1 target, Node\* curr, vector<Node\*>& result) – вспомогательный метод для метода vector<Node\*> get\_nodes\_higher(T1 val\_1);

Входные данные: дерево, ключ поиска, указатель на узел, в котором на данный момент рекурсии находится метод, вектор указателей на узлы – все узлы, ключи которых больше ключа поиска;

Формальные параметры: переменная T1 target – ключ поиска, указатель Node\* curr – указатель на узел, в котором на данный момент рекурсии находится метод, вектор vector<Node\*>& result – вектор, в котором находятся все узлы, ключи которых больше ключа поиска;

Выходные данные: вектор, в котором находятся все узлы, ключи которых больше ключа поиска.

– void \_to\_vector\_second\_value(Node\* curr, vector<T2>& result) – вспомогательный метод для метода vector<T2> to\_vector\_second\_value();

Входные данные: указатель на узел, в котором на данный момент рекурсии находится метод, вектор значений – все значения узлов дерева;

Формальные параметры: указатель Node\* curr – указатель на узел, в котором на данный момент рекурсии находится метод, вектор vector<T2>& result – вектор, в котором содержатся все значения узлов дерева;

Выходные данные: вектор, в котором содержатся все значения узлов дерева.

Класс **List** - класс, описывающий двусвязный кольцевой список со вставкой в начало.

Поля:

* Класс **Node** – класс, описывающий узел списка;
* Поля:
  + Node\* prev – указатель на предыдущий элемент;
  + Node\* next – указатель на следующий элемент;
  + T value – значение элемента списка
* Node<T>\* head – указатель на голову списка;

Методы:

– List() – конструктор списка;

Входные данные: отсутствуют;

Формальные параметры: отсутствуют;

Выходные данные: инициализированный список.

– ~List() – деструктор. Отчищает список;

Входные данные: двусвязный кольцевой список;

Формальные параметры: отсутствуют;

Выходные данные: отсутствуют.

– void add(T value) – добавляет элемент T в список в начало;

Входные данные: двусвязный кольцевой список, элемент, который добавляется;

Формальные параметры: переменная T value – элемент, который добавляется в список;

Выходные данные: двусвязный кольцевой список со вставленным элементом value.

– void Delete\_List() – вспомогательный метод для деструктора;

Входные данные: двусвязный кольцевой список;

Формальные параметры: отсутствуют;

Выходные данные: отсутствуют.

– void remove(T value) – удаляет элемент из списка;

Входные данные: двусвязный кольцевой список, элемент, который нужно удалить;

Формальные параметры: переменная T value – элемент, который удаляется из списка;

Выходные данные: двусвязный кольцевой список без элемента value.

– bool isEmpty() – возвращает true, если список пустой, иначе false;

Входные данные: двусвязный кольцевой список;

Формальные параметры: отсутствуют;

Выходные данные: true, если список пустой, иначе false.

– bool has(T value) возвращает true, если элемент есть в списке, иначе false;

Входные данные: двусвязный кольцевой список, элемент, который ищется;

Формальные параметры: переменная T value – элемент, который ищется в списке;

Выходные данные: true, если элемент есть в списке, иначе false.

– Node<T>\* find(T value) – вспомогательный метод для метода bool has(T value);

Входные данные: двусвязный кольцевой список, элемент, который ищется в списке;

Формальные параметры: переменная T value – элемент, который ищется в списке;

Выходные данные: указатель на элемент.

– void print() – выводит список в консоль;

Входные данные: двусвязный кольцевой список;

Формальные параметры: отсутствуют;

Выходные данные: все элементы списка, выведенные в консоль.

– vector<T> to\_vector() – переводит двусвязный кольцевой список в вектор. Используется для вывода дерева в окно отладки;

Входные данные: двусвязный кольцевой список;

Формальные параметры: отсутствуют;

Выходные данные: вектор значений списка.

– string to\_string(string(\*to\_str)(T&)) - превращает список в одну строчку, каждый элемент разделен «<->»;

Входные данные: двусвязный кольцевой список;

Формальные параметры: переменная string(\*to\_str) (T&) – двусвязный кольцевой список;

Выходные данные: строка string – список.

Класс **Hashtable** - класс, описывающий хэш-таблицу

Поля:

* T1 \*keys – массив всех ключей таблицы
* T2 \*values – массив всех значений таблицы
* int size – Размер массива. Всегда является степенью двойки
* int curr\_count – Текущее количество элементов в таблице
* int (\*obj\_to\_int)(const T1 &) – Функция, переводящая объект T1 в целое число
* int (\*obj\_to\_int)(const T2 &) – Функция, переводящая объект T2 в целое число

Методы:

* Hashtable(size\_type size) – конструктор, создаёт хэш-таблицу размерности size;

Входные данные: размер хеш-таблицы;

Формальные параметры: переменная size\_type size – размер хэш-таблицы;

Выходные данные: хеш-таблица размера size.

* size\_type firstHashFunction(key\_type key) – функция, высчитывающая первичную хэш-функцию;

Входные данные: ключ для хеш-функции;

Формальные параметры: переменная size\_type sum – ключ, для которого считается хэш-функция;

Выходные данные: целочисленное число – хэш-функция заданного ключа.

4.2 Описание среды разработки

Для разработки выбрана среда Visual Studio 2019, в которой использовался инструмент Windows Forms. Весь код был написан на языке C#. Windows Forms – это очень удобная среда, в которой можно задавать параметры объектов на форме, при помощи конструктора. Также, в Windows Forms уже есть более 100 различных инструментов, для организации работы с пользователем, вывода данных, и их внутренней обработки.

4.3 Руководство пользователя

При запуске программы появляется главное окно, в котором можно открыть таблицу с данными о пользователях, таблицу с данными о постах пользователей, и окно для формирования общего отчёта

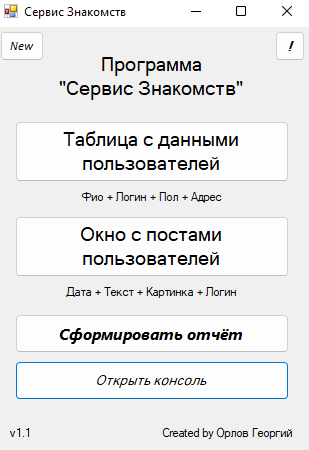


Рисунок 3 – Главное окно программы

Открыв окно «Таблица с данными пользователей», можно добавить данные о пользователи, заполнив 4 поля, найти и удалить пользователей, с указанным адресом. А также сохранить файл, или открыть новый.

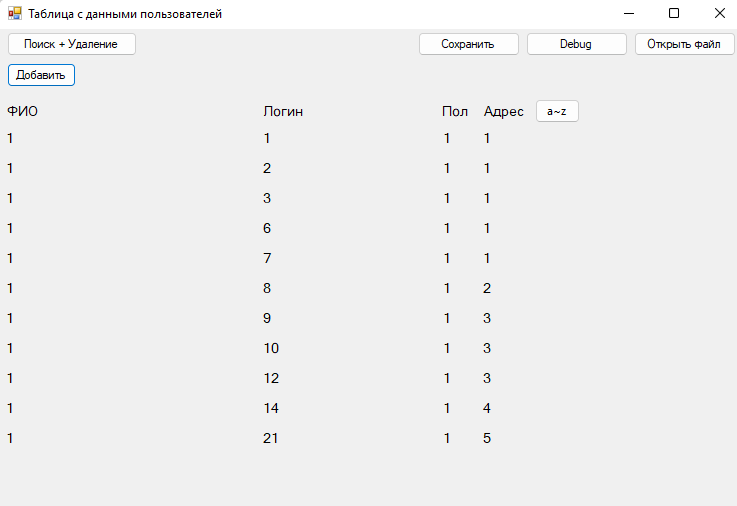


Рисунок 4 – Справочник «Пользователи»

Для добавления данных о новом пользователе нужно нажать на кнопку «Добавить», заполнить 4 поля, и нажать кнопку «Добавить», а затем кнопку «Закрыть»

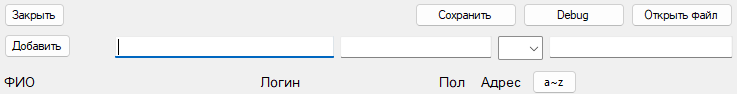


Рисунок 5 – Добавление данных о новом пользователе

Для поиска и удаления пользователей, нужно нажать на кнопку «Поиск + Удаление», ввести адрес, и нажать кнопку «Найти». У найденных записей пользователей будет кнопка «Удалить». Если нажать на неё, то пользователь будет удалён.

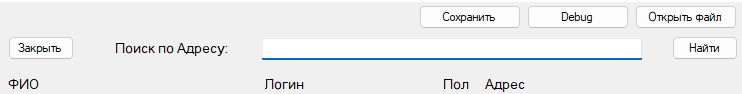


Рисунок 6 – Поиск пользователей

Открыв окно «Таблица с постами пользователей», можно добавить данные о публикациях пользователя, заполнив 4 поля, найти и удалить публикации пользователя, с указанной датой и логином. А также сохранить файл, или открыть новый.

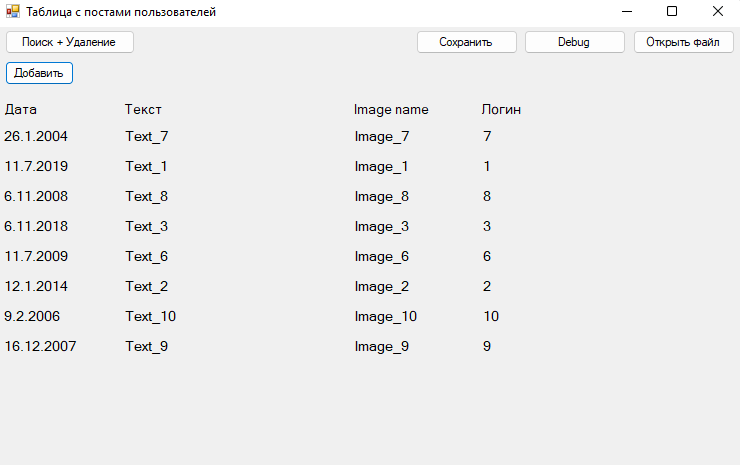


Рисунок 7 – Справочник «Публикации пользователей»

Для добавления данных о новой публикации пользователя нужно нажать на кнопку «Добавить», заполнить 4 поля, и нажать кнопку «Добавить», а затем кнопку «Закрыть»

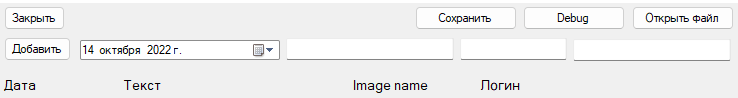


Рисунок 8 – Добавление данных о новой публикации

Для поиска и удаления пользователей, нужно нажать на кнопку «Поиск + Удаление», ввести дату и логин, и нажать кнопку «Найти». У найденных записей пользователей будет кнопка «Удалить». Если нажать на неё, то пользователь будет удалён.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 – Поиск публикаций пользователей

4.4 Тестирование

Тестирование проводилось по методу чёрного ящика. В таблицах 4, 5 описаны тестовые ситуации.

Таблица 5 – Тестирование работы со справочником «Пользователи»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| АВЛ-дерево | Атрибуты (ФИО, логин, пол, адрес) | АВЛ-дерево | Атрибуты (ФИО, логин, пол, адрес |
| Добавление | | | | | |
| 1 | Добавление некорректных данных | - | , , , , | - | Ошибка, длинна ФИО не может быть меньше 1 |
| - | 1234567890123456789  1234567890123456789 1,1,1, | - | Ошибка, длинна ФИО превышает допустимую |
| 2 | Добавление корректных данных | - | 1,1,1,1, | 1,1,1,1, | Данные успешно добавлены |
| Добавление корректных данных (Левый поворот) |  | 1,1,1,20, |  |
| Добавление корректных данных (Правый поворот) |  | 1,1,1,18, |  |
| Добавление корректных данных (Лево-Правый поворот) |  | 1,1,1,27, |  |
| Добавление корректных данных (Право-левый поворот) |  | 1,1,1,21, |  |
| Поиск | | | | | |
| 3 | Запись не существует | - | 1 | - | Запись не найдена |
| 4 | Запись существует | 1,1,1,1, | 1 | 1,1,1,1, | Запись найдена |
| Удаление | | | | | |
| 5 | Запись не существует | - | 1 | - | Запись не найдена, и не может быть удалена |
| 6 | Запись существует | 1,1,1,1, | 1 | - | Запись найдена, и успешно удалена |

Таблица 6 – Тестирование работы со справочником «Публикации пользователей»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Хэш-таблица (метод открытой адресации) | Атрибуты (дата, текст, изображение, логин) | Хэш-таблица (метод открытой адресации) | Атрибуты (дата, текст, изображение, логин) |
| Добавление | | | | | |
| 1 | Добавление некорректных данных | - | 0.0.0, , , , | - | Ошибка, логин записи не может быть меньше 1 |
| 2 | Добавление корректных данных | - | 11.11.2011,1,1,1 | 11.11.2011,1,1,1 | Запись добавлена |
| 3 | Проверка уникальности ключа | 11.11.2011,1,1,1 | 11.11.2011,1,1,1 | 11.11.2011,1,1,1 | Ошибка, публикация с такой датой и логином уже есть в нашем справочнике |
| 4 | Добавление при коллизии | 11.11.2011,1,1,1 | 11.11.2011,3,3,3 | 11.11.2011,1,1,1 11.11.2011,3,3,3 (Первичный хэш совпадает, вторичный различен) | Запись добавлена |
| Поиск | | | | | |
| 5 | Запись не существует | - | 11.11.2011,3 | - | Запись не найдена |
| 6 | Запись существует | 11.11.2011,1,1,1 | 11.11.2011,1 | - | 11.11.2011,1,1,1 |
| 7 | Поиск при коллизии | 11.11.2011,1,1,1 11.11.2011,3,3,3 | 11.11.2011,3 | - | 11.11.2011,3,3,3 |
| Удаление | | | | | |
| 8 | Запись не существует | - | 11.11.2011,1,1,1 | - | Запись не найдена |
| 9 | Запись существует | 11.11.2011,1,1,1 | 11.11.2011,1,1,1 | - | Запись удалена |
| 10 | Удаление при коллизии | 11.11.2011,1,1,1 11.11.2011,3,3,3 | 11.11.2011,3 | 11.11.2011,1,1,1 | Запись удалена |

# 

# Заключение

Целью курсового проекта было: разработка информационной системы «Сервис Знакомств».

Цель достигнута. Для достижения поставленной цели были выполнены следующие задачи:

1. Проведен анализ предметной области «Сервис Знакомств» и построена её модель;
2. Изучены теоретические основы методов построения справочников;
3. Определены требования к информационным системам;
4. Информационная система была реализована и спроектирована;
   1. Был изучен язык C# версии 9.0
   2. Изучен Windows Forms
   3. Во время разработки Windows Forms, был использован в качестве среды выполнения

# Список литературы

1. Кнут Д. Э. Искусство программирования, Том 3. Сортировка и поиск – 2-е изд. – Москва : Издательский дом «Вильямс», 2007. – 832 с.

2. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск. / Роберт Седжвик. – Киев : Издательство «ДиаСофт», 2001. – 688 с.

3. Кормен Т. Х. Алгоритмы: построение и анализ. / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, Клиффорд Штайн – 2-е изд. – Москва : Издательский дом «Вильямс», 2005. – 1296 с.