**//Тема:** Программное приложение мониторинга библиотечного фонда на //основе линейного списка

**Содержание**

[**Введение** 4](#_Toc28093691)

[**1. Теоретическая часть** 5](#_Toc28093692)

[**1.1.** **Анализ предметной области** 5](#_Toc28093693)

[**1.2.** **Выбор структур данных для реализации операций приложения** 5](#_Toc28093694)

[**1.3. Вывод к первой главе** 7](#_Toc28093695)

[**2.** **Проектная часть** 7](#_Toc28093705)

[**2.1. Общие сведения** 7](#_Toc28093696)

[**2.2. Функциональное назначение программы** 7](#_Toc28093697)

[**2.3. Описание структуры программы** 8](#_Toc28093698)

[**2.3. Алгоритмы, используемые в программе** 8](#_Toc28093699)

[**2.3. Реализация алгоритмов связного списка** 8](#_Toc28093700)

[**2.4. Реализация алгоритмов бинарного дерева** 13](#_Toc28093701)

[**2.5. Реализация заполнения связного списка из файла** 16](#_Toc28093702)

[**2.6. Проектирование** 16](#_Toc28093703)

[**2.7. Вывод ко второй главе** 22](#_Toc28093704)

[**3.** **Экспериментальная часть** 22](#_Toc28093705)

[**3.1. Тестирование** 22](#_Toc28093706)

[**3.2. Инструкция для пользователя** 24](#_Toc28093707)

[**Заключение** 25](#_Toc28093708)

[**Список используемых источников** 26](#_Toc28093709)

[**Приложение** 27](#_Toc28093710)

# **Введение**

В данной курсовой работе представлена реализация программного приложение мониторинга библиотечного фонда на основе линейного списка. В нее входят система хранения книг, система отображения всех авторов, система сортировки книг, система добавления и удаления книг из библиотеки и система очистки книг

Программная часть реализована на языке Python, с использованием библиотеки Tkinter для работы с графическим интерфейсом в среде разработки PyCharm.

Ведение учета книг в библиотеки – сложный процесс и для его упрощения, отличным решением является разработка специального приложения. Пользователь может занести или удалить информацию о книге в список, отсортировать книги по году издания, просмотреть список авторов и очистить список книг.

# **1. Теоретическая часть**

## **Анализ предметной области**

Анализируя предметную область, были выявлены данные, необходимые для заполнения пользователем.

К этим данным относятся:

* Id книги;
* Название;
* Автор книги;
* Год издания;
* Кол-во книг в библиотеки;

Также при анализе предметной области был выявлен функционал программного приложения.

К нему относится:

* Корректный ввод информации;
* Сортировка книг по году издания;

## **Выбор структур данных для реализации операций приложения**

Структура “Список” является структурой, которая даёт больше контроля над вставкой элементов в контейнер, их исключением и другими операциями. Существует две основные разновидности списков, а именно односвязные и двусвязные.

**Связный список** – это динамическая структура данных, состоящая из [узлов](http://ru.wikipedia.org/w/index.php?title=%D0%A3%D0%B7%D0%B5%D0%BB_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29&action=edit&redlink=1), каждый из которых содержит как собственные [данные](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29), так и [ссылку](http://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) на следующий узел списка. Доступ к элементу списка осуществляется только от его начала, т.е., обратной связи в этом списке нет. Просмотр односвязного списка может производиться только последовательно, начиная с начала списка. На рисунке 1.1 представлено графическое изображение линейного односвязного списка.

Преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями, также в отличии от массива в список легко добавить элемент в середину, без тяжелых или цикличных перестроений. Ниже на рисунке представлено графическое изображение линейного односвязного списка.

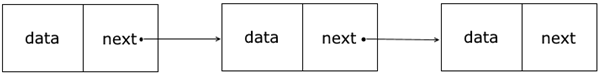


Рисунок 1.1 – Графическое изображение связного списка

**Двусвязный список** представляет из себя двусвязную динамическую структуру. В двусвязном списке каждый элемент структуры имеет два указателя, первый указывает на предыдущий элемент коллекции, а второй на последующий. Такое изменение решает проблему отсутствия возможности просмотра списка в обратном направлении.

Если элемент в списке является первым, то его указатель на предыдущий элемент указывает на None. Похожая ситуация происходит и с последним элементом коллекции, ведь его указатель на последующий элемент так же будет показывать на None. Логическая структура линейного двусвязного списка представлена на рисунке 1.2.

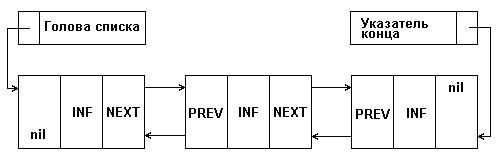


Рисунок 1.2 – Графическое изображение двусвязного списка

Для хранения информации о книгах было принято решение использовать структуру данных «связный список». Использование структуры “Односвязный список” принесёт хорошую скорость работы программы во время обработки данных

## **1.3. Вывод к первой главе**

В результате анализа предметной области, структур данных и алгоритма сортировки была построена структура всего проекта.

1. **Проектная часть**

## **2.1. Общие сведения**

Программное приложение мониторинга библиотечного фонда разработано как настольное приложение для компьютеров, с графическим пользовательским интерфейсом. Информация о книгах хранится в файле с расширением .txt.

Языком реализации проекта является Python. Также при разработке использовалась стандартная библиотека Tkinter для работы с графическим интерфейсом.

Целевой операционной системой для использования программы является Windows.

## **2.2. Функциональное назначение программы**

Данный программный продукт был создан с целью помочь работникам сферы регулирования библиотечного фонда для удобного хранения и отображения информации о книге, основные возможности программного продукта:

* Добавление информации о книге;
* Удаление информации о книге;
* Вывод информации о книге;
* Вывод списка всех авторов;
* Сортировка книг по году издания;
* Очистка списка книг;

## **2.3. Описание структуры программы**

Логическая структура программы подразумевает использование двух парадигм программирования.

Первая, объектно-ориентированный подход для удобного и наглядного представления иерархии классов и данных, хранящихся в этих классах. Данный подход непременно должен быть использован, когда речь заходит о представлении различных типов данных, как единой структуры. Ведь ООП было создано для улучшения типизации и стандартизации данных в программном коде.

Вторая, процедурный стиль программирования для различного рода дополнительных функций или операций, где производятся лишь видоизменения и преобразования над уже созданными объектами классов и переменными. ПП позволяет сохранить простоту читаемости кода, когда необходимо создать функции, внесение которых в конкретный класс неуместно, а создание нового усложняет читаемость.

## **2.3. Алгоритмы, используемые в программе**

Основными алгоритмами данного программного приложения являются операции связного списка, такие как добавление и удаление информации о книге, вывод списка авторов и другие. Также к основным алгоритмам относятся операции бинарного дерева, с помощью которых информация в связном списке сортируется.

## **2.3. Реализация алгоритмов связного списка**

**Инициализация связного списка**

Создается ссылка на головной объект и присваивается ему значение «None», так на этапе объявления в структуре присутствует только ссылка на головной элемент списка.

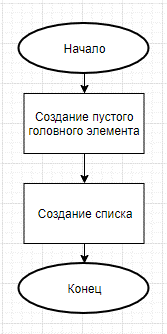


Рисунок 2.1 – Графическое изображение инициализации связного списка

**Добавление элемента**

Функция добавления «add», получает на вход информацию о книге. Следующим шагом она проверяет список на наличие головы, при ее отсутствии вызывается генератор узла, головному элементу присваивается новый узел, а ссылка на следующий элемент равняется «None». В случае, когда список уже имеет один или более узлов, функция обходит весь список и находит последний, далее генерируется новый узел и добавляется в конец.

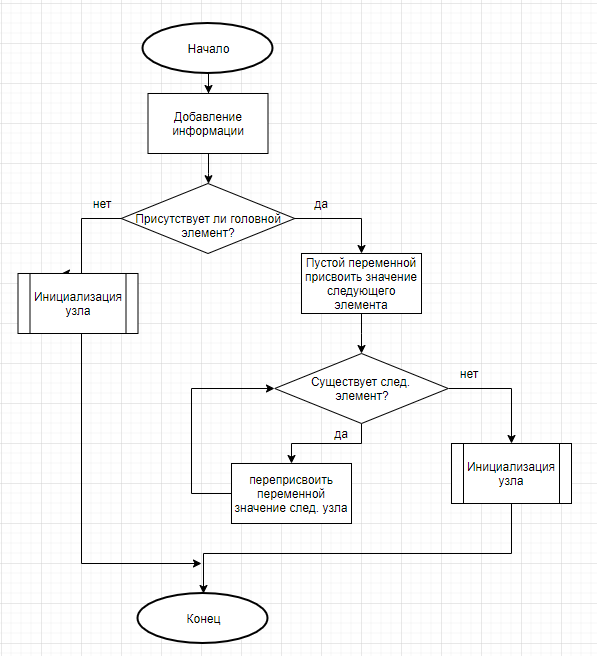


Рисунок 2.2 – Графическое изображение добавления элемента в связный список

**Удаление элемента**

Функция удаления «del\_element», получает на вход значение ключа, который необходимо удалить. Следующим шагом она проверяет список на наличие элемента с таким ключом, при отсутствие такого возвращается значение «None». В случае, когда ключ найден берется предыдущий узел и его ссылка меняется: теперь его ссылка указывает на узел следующий после того, который необходимо удалить. Алгоритм прохода: в цикле мы проходимся по всему списку пока не найдем элемент с ссылкой на узел, равной ключу. При этом важно «вести» при проходе 2 переменные, хранящие в себе предыдущий и последующий элементы. Это позволить легко изменить ссылки при удалении.

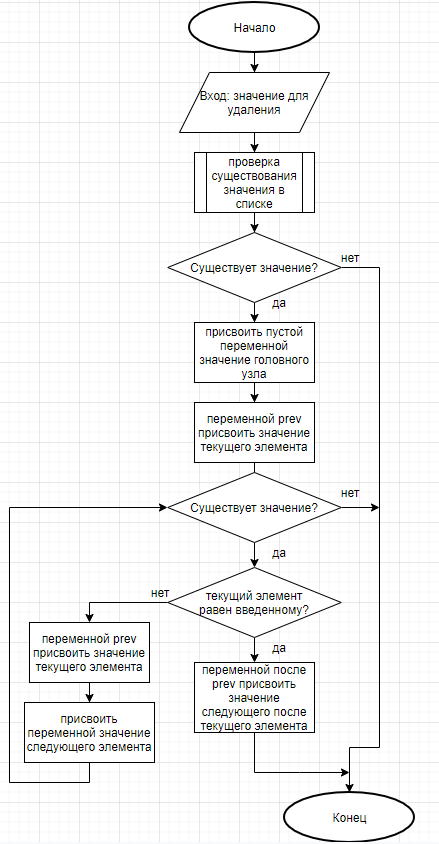


Рисунок 2.3 – Графическое изображение удаления элемента из списка

**Вывод связного списка**

Функция вывода «print» проверяет список на наличие головного элемента, при его отсутствии возвращается значение «None». В случае, когда головной элемент присутствует, переменной current присваивается его значение, и выводится на экран, пока следующий элемент списка не равен «None», переменной current присваивается значение этого элемента и выводится на экран.

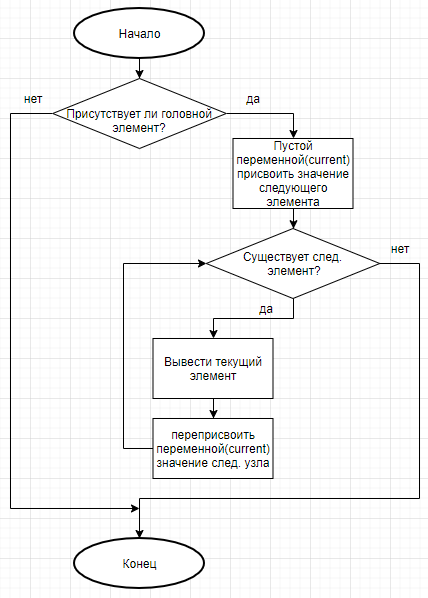


Рисунок 2.4 – Графическое изображение вывода связного списка

**Заполнение связного списка из текстового файла**

Данные о книгах в библиотеке хранятся в текстовом файле(.txt). Для заполнения связного списка информацией из файла необходимо нажать на кнопку «Авто заполнение».

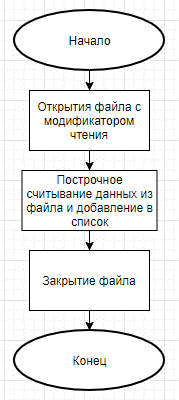


Рисунок 2.5 – Графическое изображение заполнения связного списка

## **2.4. Реализация алгоритмов бинарного дерева**

Поскольку информация о книгах в библиотеке может располагаться беспорядочно, для более удобного просмотра списка книг, их можно отсортировать по году издания.

В качестве структуры для сортировки информации была выбрана сортировка бинарным деревом.

**Бинарное дерево** — это иерархическая структура данных, в которой каждый узел имеет значение (ключ) и ссылки на левого и\или правого потомка. Ниже на рисунке представлено графическое изображение бинарного дерева.

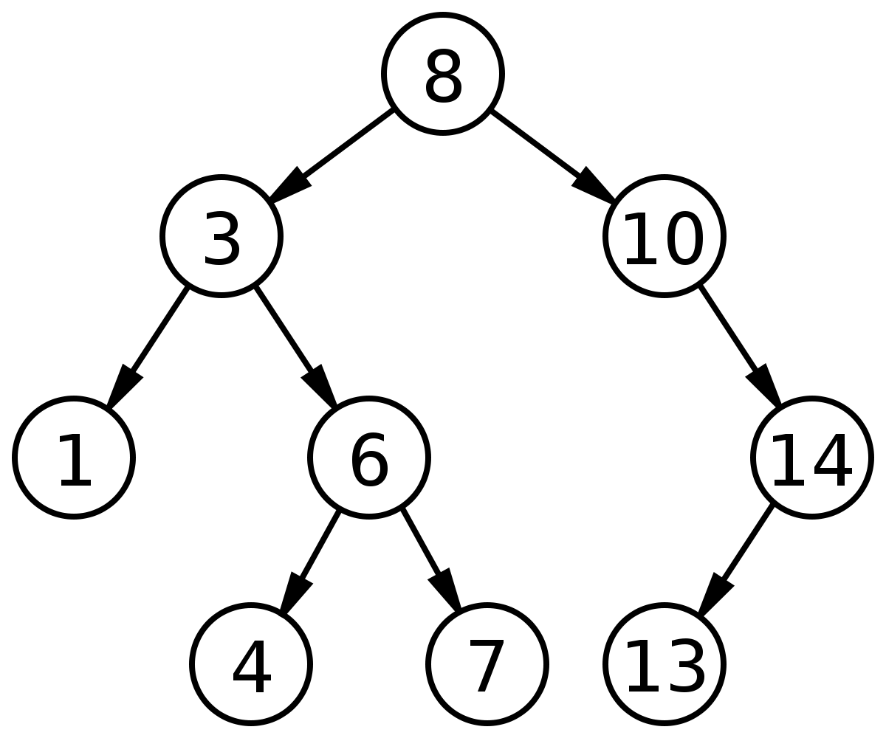


Рисунок 2.6 – Графическое изображение бинарного дерева

**Добавление элемента в бинарное дерево**

Рекурсивная функция, в случае если дерево не существует или веток больше нет, элемент добавляется на текущую позицию и происходит выход из рекурсии. В случае если добавляемый элемент меньше текущего, выбирается левая ветка дерева, иначе выбирается правая ветка дерева.

Узел – элемент дерева, имеющий некоторое значение и ссылку на левого и\или правого потомка. Узлы, не имеющие потомков, называется листьями.

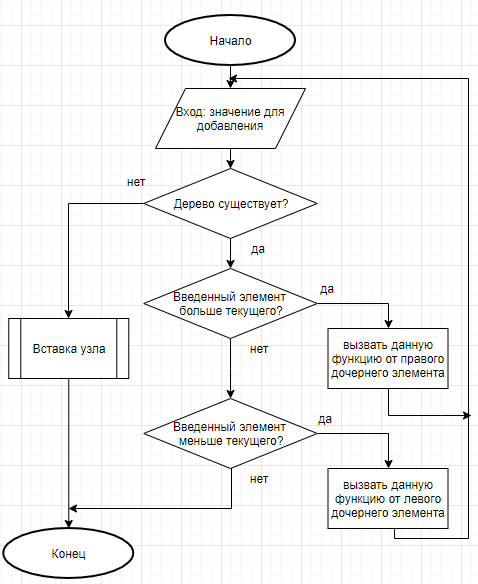


Рисунок 2.7 – Графическое изображение добавление элемента в дерево

**Сортировка элементов в бинарном дереве**

Рекурсивная функция, в случае если дерево не существует сортировка будет остановлена, иначе, пока существует левого поддерево, функция будет вызываться с передачей левого поддерева в качестве параметра, после сортировки левого поддерева, функция будет вызываться с передачей правого поддерева в качестве параметра пока оно существует.

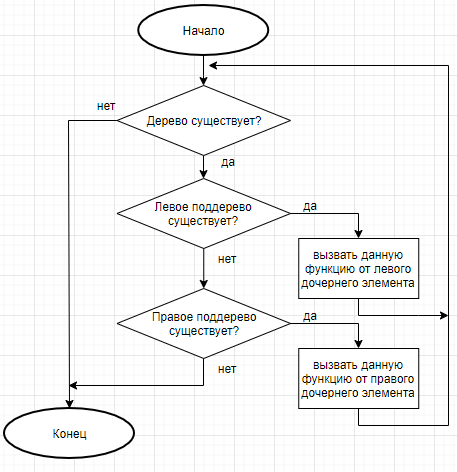


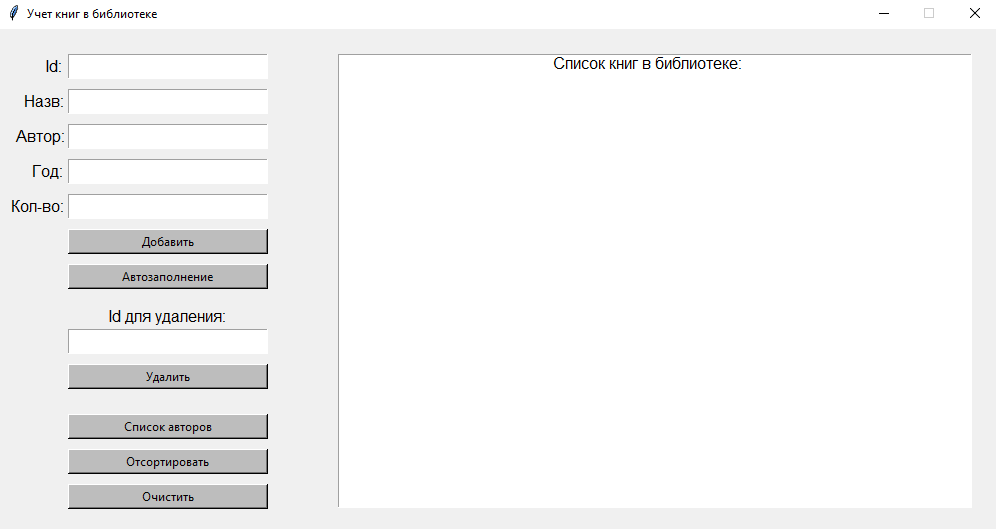
Рисунок 2.8 – Сортировка элементов в бинарном дереве

## **2.5. Реализация заполнения связного списка из файла**

Данные о книгах в библиотеке хранятся в текстовом файле(.txt). Для заполнения связного списка информацией из файла необходимо нажать на кнопку «Авто заполнение».

## **2.6. Проектирование**

Всего приложение содержит одно окно. Графический интерфейс проектировался с целью удобства и понятности использования. Пользовательский интерфейс состоит из пяти полей ввода для заполнения информации о книге, поля ввода id для удаления, пяти кнопок, реализующих функции добавления, удаления, заполнения, сортировки книг, очистки библиотеки и вывода списка авторов.

Рисунок 2.9 – Пользовательский интерфейс приложения

У каждого поля свой тип данных, и что бы программа не прекращала работу при некорректном вводе, будет обработана и выведена ошибка, что введенные данные некорректны, и вы сможете повторить ввод.

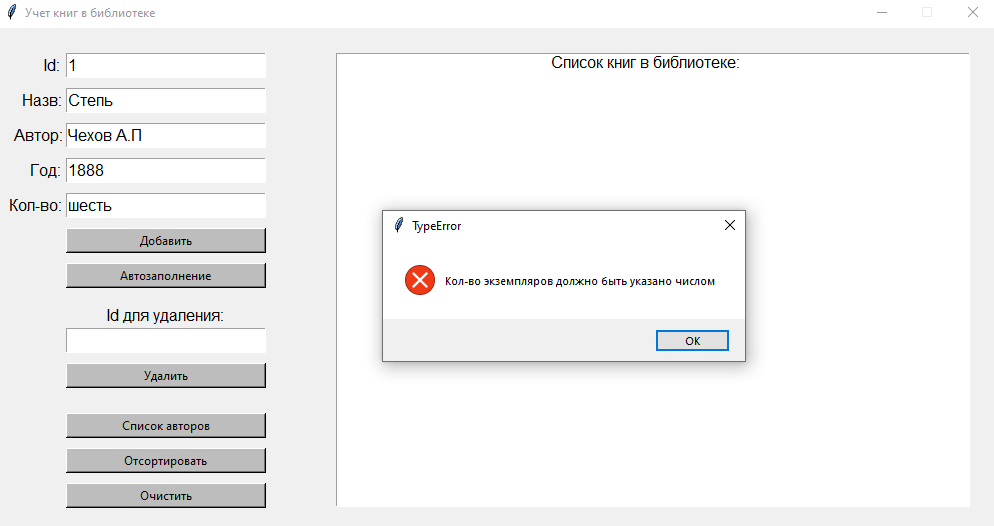


Рисунок 2.10 – Окно ошибки при некорректном вводе кол-ва экземпляров

При добавлении информации о новой книге необходимо заполнить все поля, в случае если этого не сделать будет выведена соответствующая ошибка.

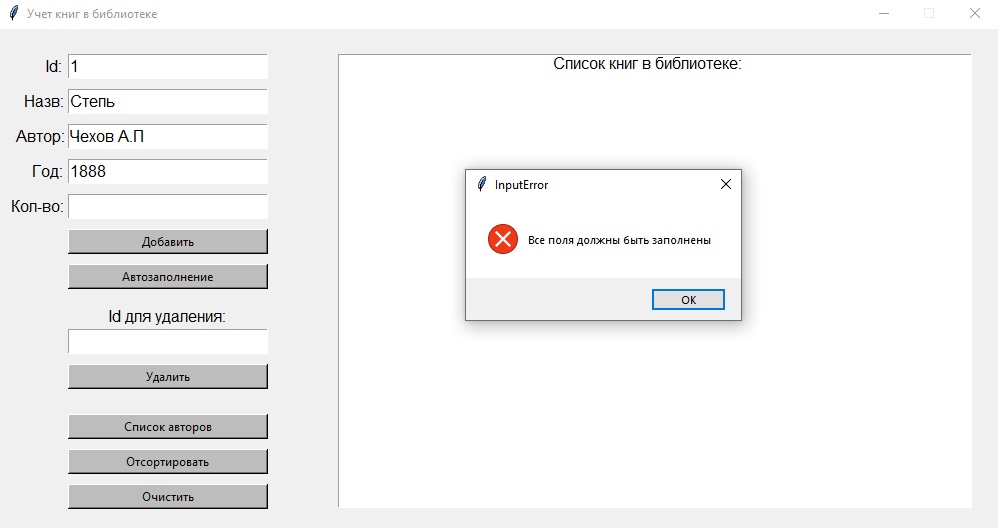


Рисунок 2.11 – Окно ошибки при неполном заполнении

Если все поля заполнены корректно, информация о книге будет добавлена в список и выведена в окно приложения.

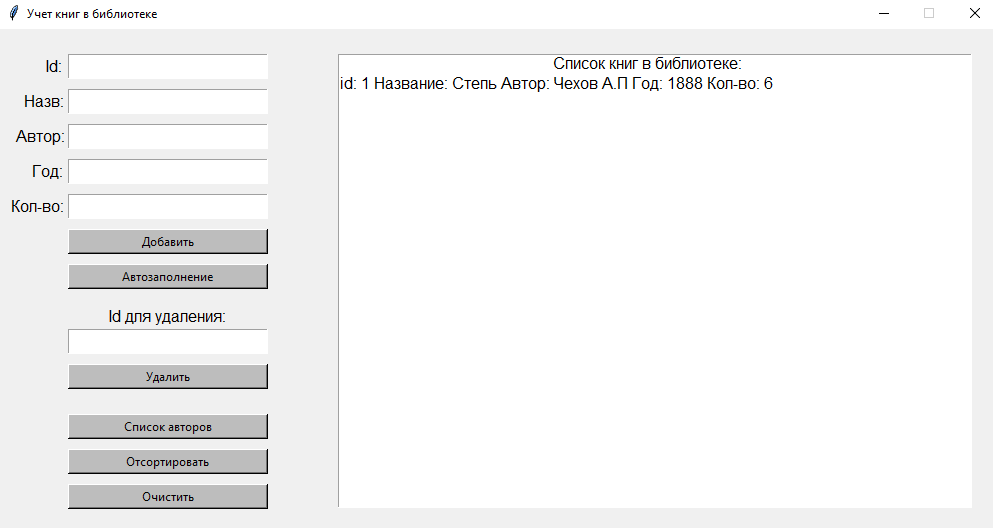


Рисунок 2.12 – Корректное добавление информации о книге

При попытке добавления в список книги с id, которое уже находится в списке, будет выведена ошибка.

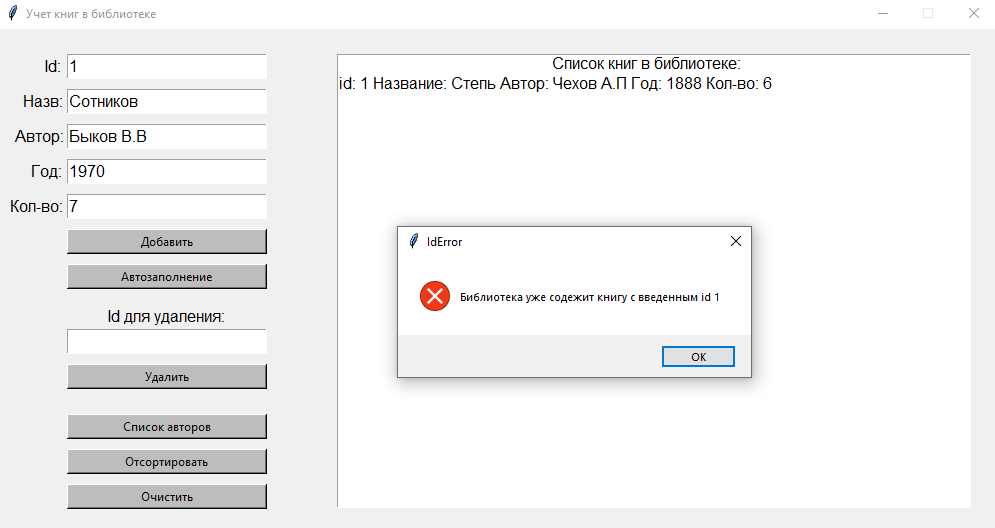


Рисунок 2.13 – Окно ошибки при добавлении повторного id

При попытке удаления книги, записав id которого не существует, будет выведена ошибка.

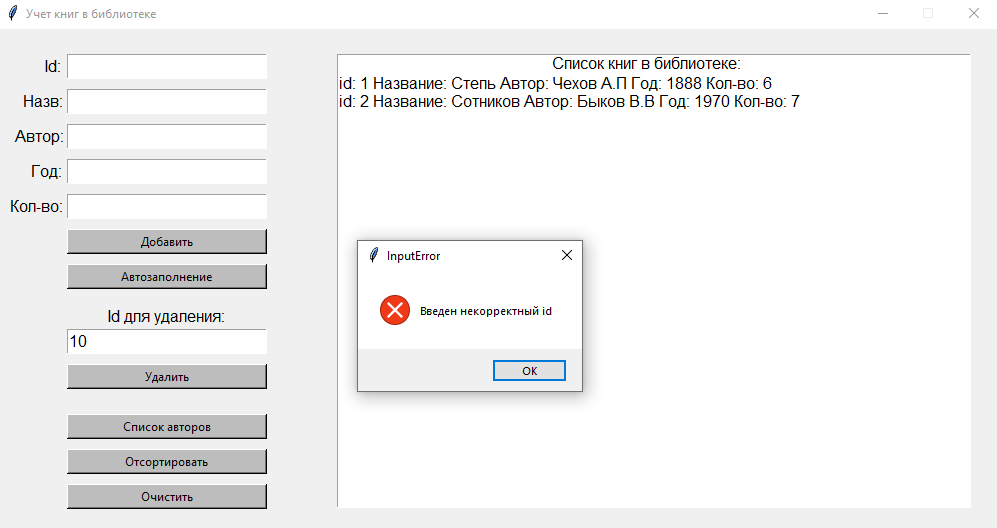


Рисунок 2.14 – Окно ошибки при некорректном вводе id для удаления

Если необходимо заполнить список книгами из файла, необходимо нажать на кнопку «Автозаполнение». После нажатия на кнопку, список будет очищен во избежание проблем с повторяющими id и заполнен информацией из текстового файла.

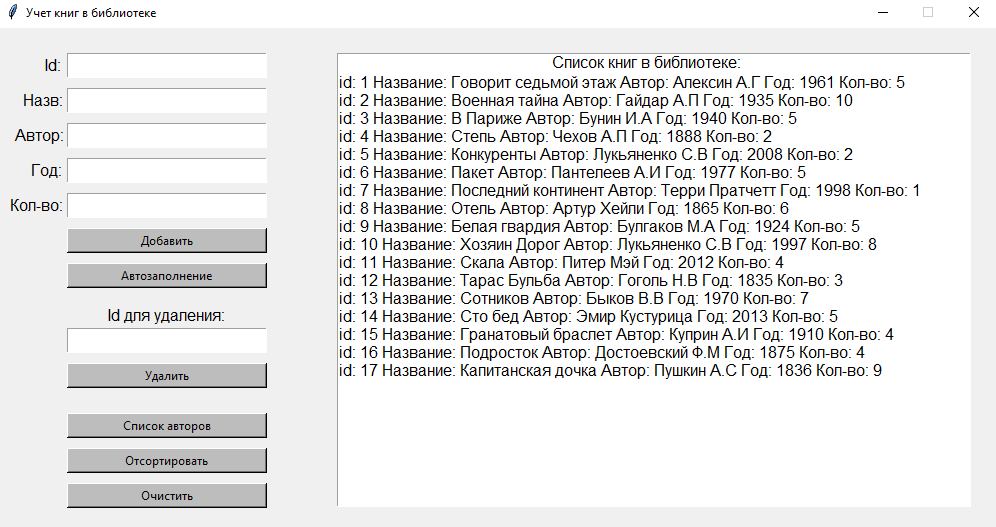


Рисунок 2.15 – Заполнение списка информацией из файла

В случае если пользователю необходимо просмотреть список авторов, чьи книги присутствуют в библиотеке, необходимо нажать на кнопку «Список авторов» и в правой части окна появится список авторов.

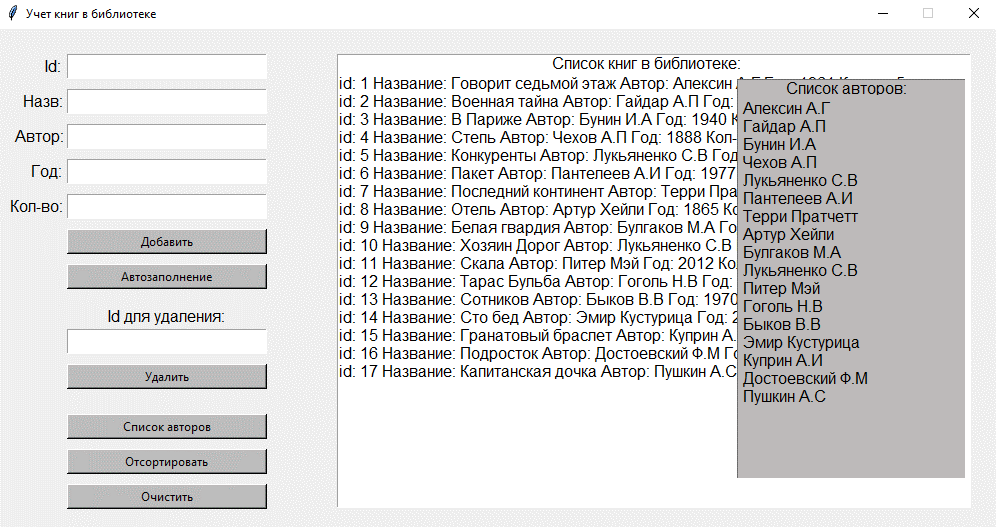


Рисунок 2.16 – Просмотр списка авторов книг

Для более удобного просмотра содержимого библиотеки, программное приложение содержит функцию сортировки, для ее использования необходимо нажать на кнопку «Отсортировать».

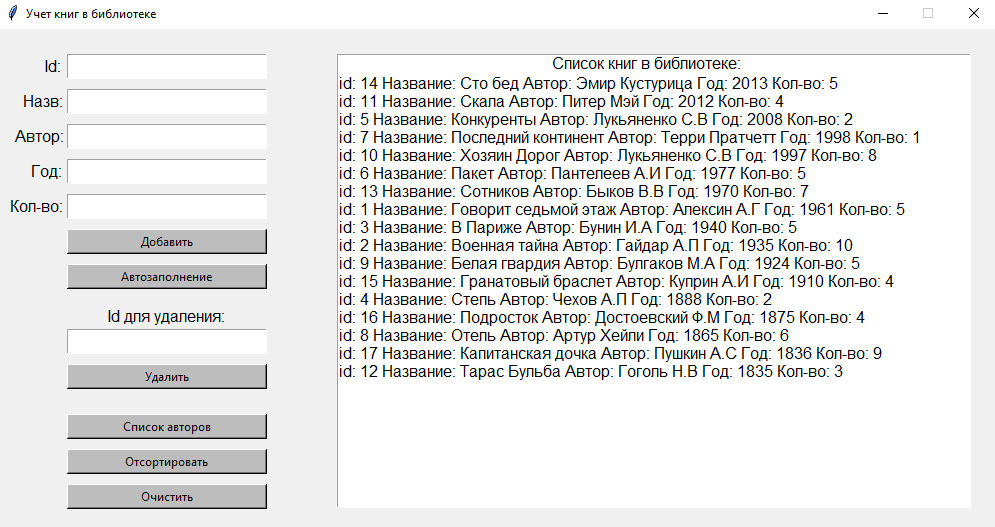


Рисунок 2.17 – Просмотр отсортированного содержимого списка

При необходимости удаления информации о всех книгах в библиотеке, необходимо нажать на кнопку «Очистить» и список книг будет очищен.

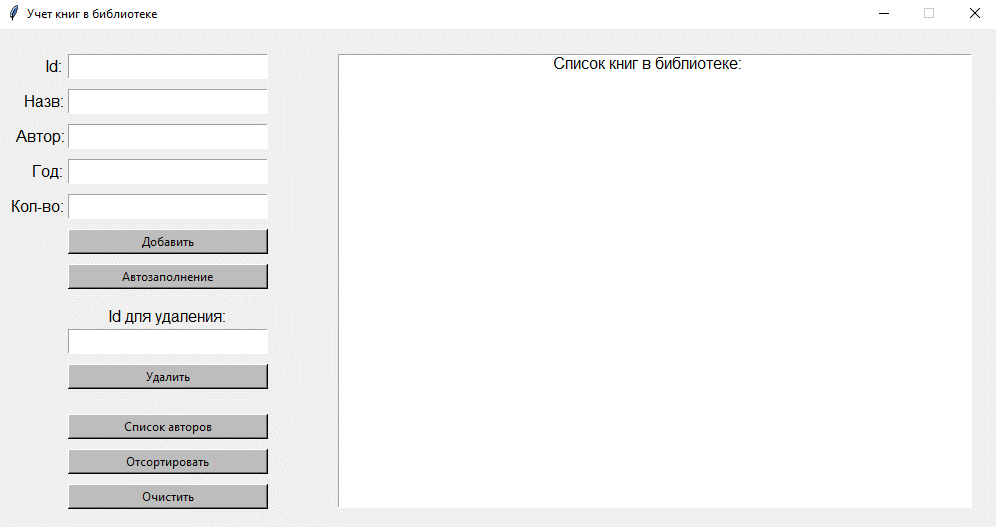


Рисунок 2.18 – Очистка списка книг

## **2.7. Вывод ко второй главе**

В результате работы над проектной частью был спроектирован пользовательский интерфейс проекта.

# **Экспериментальная часть**

## **3.1. Тестирование**

Целью проведения испытаний является:

* проверка работоспособности функций программного продукта;
* проверка надежности функционирования программного продукта;
* проверка соответствия заявленным характеристикам и требованиям, изложенным в техническом задании;

|  |  |  |  |
| --- | --- | --- | --- |
| Входные  данные | Ожидаемый результат | Фактический результат | Статус теста |
| Пустые поля ввода | Сообщение о том, что все поля должны быть заполнены | Сообщение о том, что все поля должны быть заполнены | Пройден |
| 1, Сотников, Быков В.В, 1970, 7 | Добавление информации о книге в список | Добавление информации о книге в список | Пройден |
| 2, Белая гвардия, Булгаков М.А, 1924, 5 | Добавление информации о книге в список | Добавление информации о книге в список | Пройден |
| Ввод id, которого нет в библиотеке в поле для удаления | Сообщение о том, что введен некорректный id | Сообщение о том, что введен некорректный id | Пройден |
| 3, Конкуренты, Лукьяненко С.В, 2008, 2 | Добавление информации о книге в список | Добавление информации о книге в список | Пройден |
| 3, Капитанская дочка, Пушкин А.С, 1836, 9 | Сообщение о том, что введенный id уже содержится в библиотеке | Сообщение о том, что введенный id уже содержится в библиотеке | Пройден |
| Ввод 3 в поле для удаления | Удаление книги с введенным id | Удаление книги с введенным id | Пройден |
| Три, Гранатовый браслет, Куприн А.И, 1910, 4 | Сообщение о том, что id должен быть указан числом | Сообщение о том, что id должен быть указан числом | Пройден |

Тестовые запуски показали, что программное приложение:

* Работоспособно;
* Устойчиво к ошибкам пользователя;
* Интуитивно понятно, благодаря простому интерфейсу программы;
* Приложение выполнено полностью в соответствии с техническим заданием.

## **3.2. Инструкция для пользователя**

**Добавление книги:**

1. Заполнить поля необходимые для добавления. Поля которые требуется заполнять показаны на рисунке 3.1.

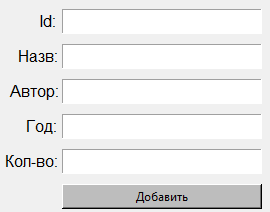


Рисунок 3.1 — Поля заполнения для добавления книги

1. Нажать на кнопку «Добавить». После добавления книги, она будет выведена в список, находящийся в правой части окна, что изображено на рисунке 3.2.

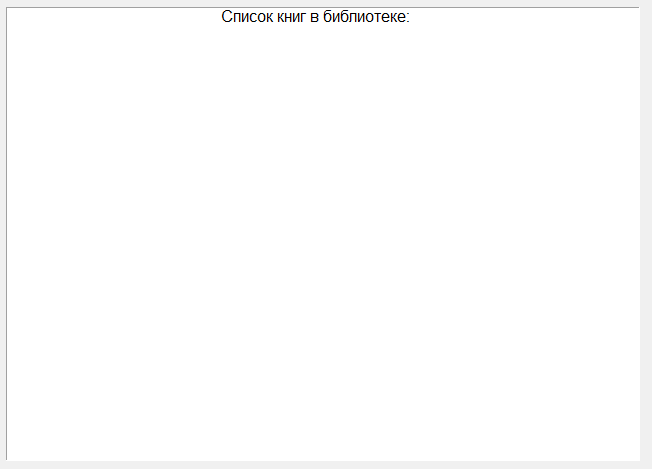


Рисунок 3.2 — Окно вывода списка книг

**Удаление книги:**

1. Ввести id книги, которую требуется удалить в поле, показанное на рисунке 3.3.
2. Нажать на кнопку «Удалить».

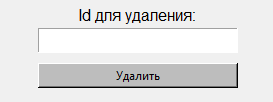


Рисунок 3.3 — Поле заполнения для удаления книги

**Сортировка списка книг:**

1. Нажать на кнопку «Отсортировать», которая показана на рисунке 3.4.



Рисунок 3.4 — Кнопка сортировки списка книг

**Очистка списка книг:**

1. Нажать на кнопку «Очистить», которая показана на рисунке 3.5.



Рисунок 3.5 — Кнопка очистки списка книг

**Заполнение списка книг из файла:**

1. Нажать на кнопку «Автозаполнение», которая показана на рисунке 3.6.



Рисунок 3.6 — Кнопка автоматического заполнения списка книг

1. После заполнения списка, он будет выведен в список, находящийся в правой части окна, что показано на рисунке 3.7.



Рисунок 3.7 — Окно вывода списка книг

# **Заключение**

В результате выполнения курсовой работы мы имеем протестированное и исправно работающее программное приложение мониторинга библиотечного фонда на основе линейного списка.

Реализованное программное приложение имеет следующие функциональные возможности:

1) Добавления информации о книге;

2) Реализована система для контроля ввода;

3) Сортировка книг в библиотеке по году издания;

4) Вывод данных из файла в список;

5) Вывод списка авторов;

6) Удаление информации о книге;

7) Очистка списка книг.

В процессе работы над курсовой работой были закреплены навыки:

* Анализа предметной области;
* Работы с текстовыми файлами.
* Разработки на языке Python;
* Проектирования графического интерфейса пользователя c помощью стандартной библиотеки Tkinter;
* Тестирования программного продукта;
* Подготовки документации.

# **Список используемых источников**

Нормативные документы:

1. ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: Стандартинформ, 2017. – 32 с.;

1. О введении в действие Инструкции по организации и проведению курсового проектирования. – М.: РТУ МИРЭА, Приказ №1325 от 05.10.2018. – 17 с.;
2. Двоичное дерево [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/Двоичное_дерево>
3. Связный список [Электронный ресурс]. – URL:

<https://ru.wikipedia.org/wiki/Связный_список>

1. Руководство по Tkinter [Электронный ресурс]. – URL:

https://docs.python.org/3/library/tkinter.html

1. Построение блок-схем [Электронный ресурс]. – URL:

https://www.draw.io

# **Приложение**

Library.py

import tkinter as tk

from tkinter import messagebox

from Tree import \*

from List import \*

Data\_List = [] # список дат

Sorted\_list = [] # временный список для сортировки

root = tk.Tk()

text\_list = tk.Text(root, font=12, width=70, height=25)

text\_list.configure(state="disabled")

text\_list.place(relx=0.34, rely=0.05)

text\_authors = tk.Text(root, font=12, width=25, height=22.35, bg='#BDBABA')

text\_authors.configure(state="disabled")

text\_authors.place()

text\_authors.place\_forget()

def insert(tree, data): # функция вставки элемента в дерево

if not tree:

tree = TreeNode(data)

elif data < tree.data:

tree = TreeNode(tree.data, insert(tree.left, data), tree.right)

elif data > tree.data:

tree = TreeNode(tree.data, tree.left, insert(tree.right, data))

return tree

class LinkedList: # класс односвязного списка

def \_\_init\_\_(self):

self.head = None

self.length = 0

def clear(self): # функция очистки односвязного списка

self.\_\_init\_\_()

def len(self): # функция подсчета размена односвязного списка

self.length = 0

if self.head is not None:

self.length -= -1

cur = self.head

while cur.next is not None: # перебор элементов для подсчета длины

cur = cur.next

self.length -= -1 # подсчет длины односвязного списка

return self.length

def add(self, data): # функция добавления элемента в односвязный список

split\_book = data.split(' : ')

if not split\_book[3].isdigit(): # проверка правильности введенных данных

messagebox.showerror("TypeError", "Год издания должен быть указан числом")

return

if not split\_book[4].isdigit(): # проверка правильности введенных данных

messagebox.showerror("TypeError", "Кол-во экземпляров должно быть указано числом")

return

if not split\_book[0].isdigit(): # проверка правильности введенных данных

messagebox.showerror("TypeError", "Id должен быть указан числом")

return

cur = self.head

for i in range(list2.len()):

while cur is not None: # перебор элементов для проверки, существует ли введенный id в односвязном списке

temp1 = cur.value.split(' : ')

if split\_book[0] == temp1[0]:

messagebox.showerror("IdError", "Библиотека уже содежит книгу с введенным id " + split\_book[0])

return

cur = cur.next

NewNode = ListNode(data)

NewNode.next = self.head

self.head = NewNode

def del\_element(self, index): # функция удаления элемента из односвязного списка

if not str(index).isdigit(): # проверка правильности введенных данных

messagebox.showerror("TypeError", "Id для удаления должен быть указан числом")

return

text\_authors.place\_forget()

label\_authors.place\_forget()

cur = self.head

temp2 = cur.value.split(' : ')

if temp2[0] == index:

self.head = cur.next

return

can = False

while cur is not None: # перебор элементов списка для нахождения индекса в списке

temp1 = cur.value.split(' : ')

if temp1[0] == index: # проверка наличия индекса в односвязном списке

can = True

break

cur = cur.next

if can: # если индекс существует в односвязном списке, то происходит его удаление

current = self.head

while current is not None:

temp2 = current.value.split(' : ')

if temp2[0] == index:

break

prev = current

current = current.next

if current is None:

return

prev.next = current.next

else:

messagebox.showerror("InputError", "Введен некорректный id")

def print(self): # функция вывода односвязного списка на форму

text\_authors.place\_forget()

label\_authors.place\_forget()

text\_list.configure(state="normal")

text\_list.delete(1.0, tk.END)

cur = self.head

while cur is not None: # перебор элементов односвязного списка для вывода на форму

tmp = str(cur.value).split(" : ")

text\_list.insert(0.0, "id: " + tmp[0] + " Название: " + tmp[1] + " Автор: " + tmp[2] + " Год: " + tmp[3] +

" Кол-во: " + tmp[4])

text\_list.insert(0.0, '\n')

cur = cur.next

text\_list.configure(state="disabled")

def author(self): # функция вывода списка авторов

cur = self.head

text\_authors.configure(state="normal")

text\_authors.delete(1.0, tk.END)

text\_authors.place(relx=0.74, rely=0.1)

label\_authors.place(relx=0.75, rely=0.101, relwidth=0.20, relheight=0.03)

while cur is not None: # перебор элементов односвязного списка для вывода списка авторов на форму

temp1 = cur.value.split(' : ')

text\_authors.insert(0.0, " " + temp1[2])

text\_authors.insert(0.0, '\n')

cur = cur.next

text\_authors.configure(state="disabled")

def tree\_sort(self, list): # функция сортировки с помощью двоичного дерева

list.clear()

cur = self.head

while cur is not None:

list.append(cur.value)

cur = cur.next

def autoFill():

with open('lib.txt', encoding="utf8") as f: # заполнение библиотеки из файла

list1 = f.read().splitlines()

text\_list.configure(state="normal")

for el in range(len(list1)):

list2.add(list1[el])

list1.reverse()

for ele in range(len(list1)): # вывод списка книг на форму

tmp1 = str(list1[ele]).split(" : ")

text\_list.insert(0.0, "id: " + tmp1[0] + " Название: " + tmp1[1] + " Автор: " + tmp1[2] + " Год: " + tmp1[

3] + " Кол-во: " + tmp1[4])

text\_list.insert(0.0, '\n')

text\_list.configure(state="disabled")

def tree\_builder(): # функция постоения дерева

text\_authors.place\_forget()

label\_authors.place\_forget()

tree = None

list2.tree\_sort(list1)

for l in range(len(list1)): # формирование дерева

split\_tree = str(list1[l]).split(' : ')

Data\_List.append(split\_tree[3])

tree = insert(tree, Data\_List[l])

sorted\_tree(tree) # вызов функции сортировки

def sorted\_tree(tree): # функция сортировки linked list

for l in range(len(Data\_List)):

for j in range(len(Data\_List)):

try:

if sort(tree)[l] == Data\_List[j]: # сравнение каждого элемента дерева с каждым элементом списка дат

Sorted\_list.append(list1[j]) # формирование буферного отсортированного списка

except IndexError:

pass

list2.clear()

for l in range(len(Sorted\_list)):

list2.add(Sorted\_list[l]) # заполнение linked list элементами буферного списка

list2.print()

Data\_List.clear()

Sorted\_list.clear()

def sort(tree): # рекурсивная функция сортировки дерева

if not tree:

return []

return sort(tree.right) + [tree.data] + sort(tree.left)

def add\_book(): # функция добавления книги в библиотеку

text\_authors.place\_forget()

label\_authors.place\_forget()

if len(entry\_id.get()) != 0 and len(entry\_title.get()) != 0 and len(entry\_author.get()) != 0 and \

len(entry\_year.get()) != 0 and len(entry\_count.get()) != 0:

buff = entry\_id.get() + " : "

buff += entry\_title.get() + " : "

buff += entry\_author.get() + " : "

buff += entry\_year.get() + " : "

buff += entry\_count.get()

list2.add(buff) # добавление книги в linked list

else:

messagebox.showerror("InputError", "Все поля должны быть заполнены")

list2.print()

list1 = []

list2 = LinkedList()

# region UI создание графического интерфейса

label\_name = tk.Label(root, font=12, text="Список книг в библиотеке:", fg='black', bg='white')

label\_name.place(relx=0.4, rely=0.051, relwidth=0.50, relheight=0.03)

button\_add = tk.Button(root, text="Добавить", bg='#2E8B57', command=lambda: (add\_book(), list2.print()))

button\_add.place(relx=0.07, rely=0.40, relwidth=0.20, relheight=0.05)

button\_add = tk.Button(root, text="Автозаполнение", bg='#2E8B57', command=lambda: (list2.clear(), autoFill(), list2.print()))

button\_add.place(relx=0.07, rely=0.47, relwidth=0.20, relheight=0.05)

button\_print = tk.Button(root, text="Очистить", bg='#8A0808', command=lambda: (list2.clear(), list2.print()))

button\_print.place(relx=0.07, rely=0.91, relwidth=0.20, relheight=0.05)

button\_sort = tk.Button(root, text="Отсортировать", bg='#2E8B57', command=lambda: (tree\_builder(), list2.print()))

button\_sort.place(relx=0.07, rely=0.84, relwidth=0.20, relheight=0.05)

button\_del = tk.Button(root, text="Удалить", bg='#8A0808', command=lambda: (list2.del\_element(entry\_del.get()),

list2.print()))

button\_del.place(relx=0.07, rely=0.67, relwidth=0.20, relheight=0.05)

button\_authors = tk.Button(root, text="Список авторов", bg='#2E8B57', command=lambda: list2.author())

button\_authors.place(relx=0.07, rely=0.77, relwidth=0.20, relheight=0.05)

entry\_id = tk.Entry(root, font=12)

entry\_id.place(relx=0.07, rely=0.05, relwidth=0.20, relheight=0.05)

entry\_title = tk.Entry(root, font=12)

entry\_title.place(relx=0.07, rely=0.12, relwidth=0.20, relheight=0.05)

entry\_author = tk.Entry(root, font=12)

entry\_author.place(relx=0.07, rely=0.19, relwidth=0.20, relheight=0.05)

entry\_year = tk.Entry(root, font=12)

entry\_year.place(relx=0.07, rely=0.26, relwidth=0.20, relheight=0.05)

entry\_count = tk.Entry(root, font=12)

entry\_count.place(relx=0.07, rely=0.33, relwidth=0.20, relheight=0.05)

entry\_del = tk.Entry(root, font=12)

entry\_del.place(relx=0.07, rely=0.6, relwidth=0.20, relheight=0.05)

label\_id = tk.Label(root, font=12, text="Id:", fg='black')

label\_id.place(relx=0.045, rely=0.05)

label\_title = tk.Label(root, font=12, text="Назв:", fg='black')

label\_title.place(relx=0.023, rely=0.12)

label\_author = tk.Label(root, font=12, text="Автор:", fg='black')

label\_author.place(relx=0.015, rely=0.19)

label\_year = tk.Label(root, font=12, text="Год:", fg='black')

label\_year.place(relx=0.031, rely=0.26)

label\_count = tk.Label(root, font=12, text="Кол-во:", fg='black')

label\_count.place(relx=0.01, rely=0.33)

label\_del\_id = tk.Label(root, font=12, text="Id для удаления:", fg='black')

label\_del\_id.place(relx=0.07, rely=0.55, relwidth=0.20, relheight=0.05)

label\_authors = tk.Label(root, font=12, text="Список авторов:", fg='black', bg='#BDBABA')

label\_authors.place()

label\_authors.place\_forget()

# endregion

if \_\_name\_\_ == "\_\_main\_\_":

root.title("Учет книг в библиотеке")

root.geometry("1000x500")

root.resizable(False, False)

root.mainloop()

book.py

class Book(object):

id = None

title = None

author = None

year = None

count = None

def \_\_init\_\_(self, id, title, author, year, count):

self.id = id

self.title = title

self.author = author

self.year = year

self.count = count

Tree.py

class TreeNode: # класс дерева

def \_\_init\_\_(self, data, left=None, right=None):

self.data = data

self.left = left

self.right = right

List.py

class ListNode: # класс ячейки

def \_\_init\_\_(self, value=None):

self.value = value

self.next = None