САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЛЕСОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ С.М. КИРОВА

Кафедра информационных систем и технологий

Курсовая работа

По дисциплине «Технологии обработки информации»

Разработка приложения «Home Library»

Выполнил:
Студент ИЛиП, 2 курс,
группа зЛПБ-ИСиТ-19-1,
№ зачётной книжки 319014,
Галкин А.А.
Проверил:
Колмогорова С.С.

Санкт-Петербург
2021 г.

Оглавление

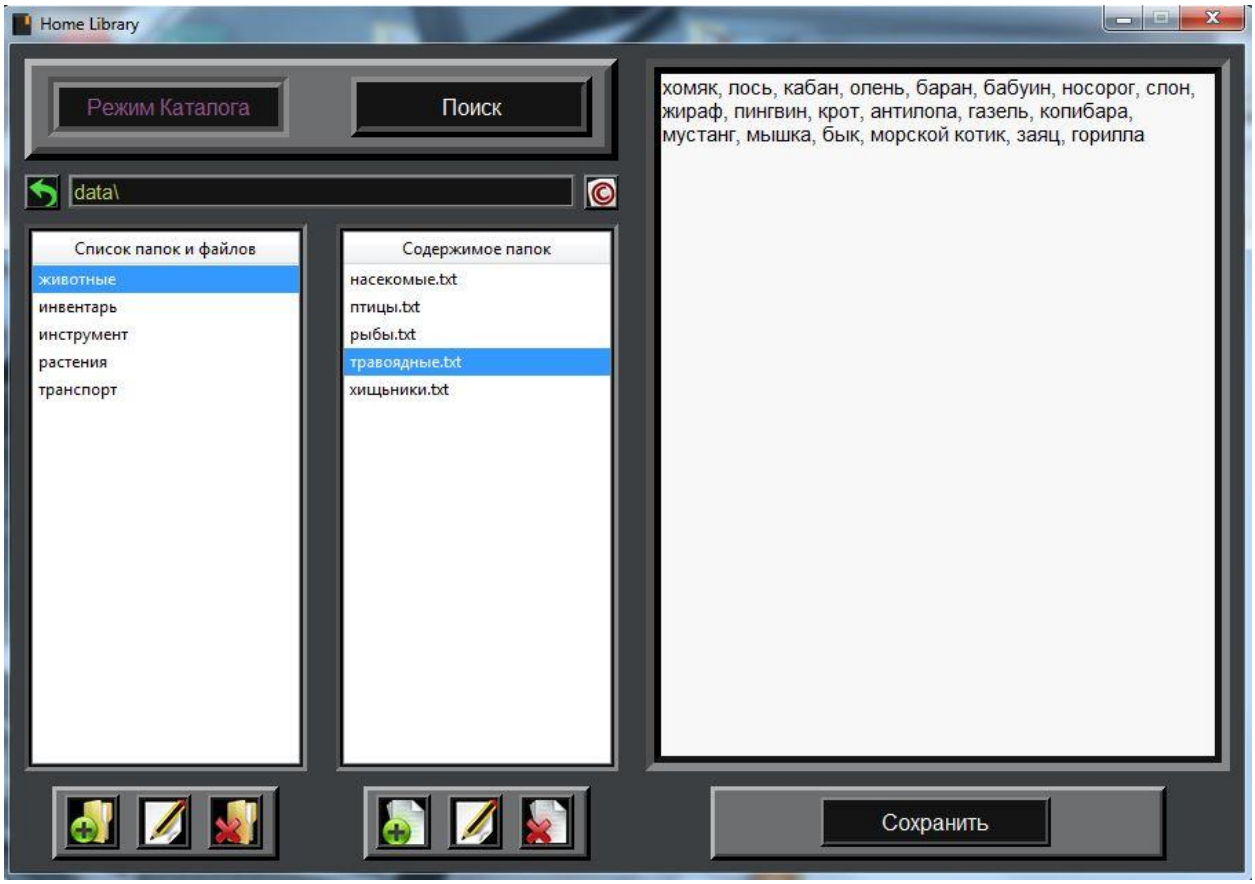
Постановка задачи

Разработать приложение, которое позволяет пользователю просматривать и формировать список папок (включать или исключать папки из списка). Для сформированного списка папок приложение формирует полный список текстовых файлов (файлов с расширением «*.txt»). Формируемый список должен включать полные имена файлов.

По команде пользователя приложение формирует индекс для списка ключевых слов для всех файлов. Список ключевых слов должен храниться в файле «KeyList.txt», который должен находиться в одной папке с приложением. Формат файла: одна строка – одно ключевое слов. Приложение читает этот файл при старте. Индекс формируется по точному соответствию (за исключением регистра) ключевым словам. После того, как индекс сформирован, пользователь указывает (выделяет в списке) ключевое слово и выполняет поиск файлов, в которых встречается это ключевое слово. Двойной щелчок на имени найденного файла должен открывать окно с этим файлом и выделенным ключевым словом ближайшим к началу файла.

Интерфейс приложения

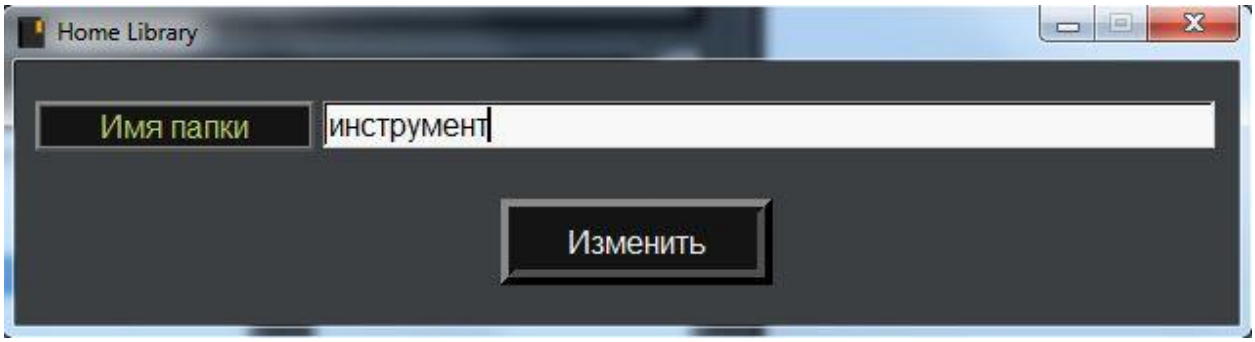
Главное окно программы выглядит следующим образом:

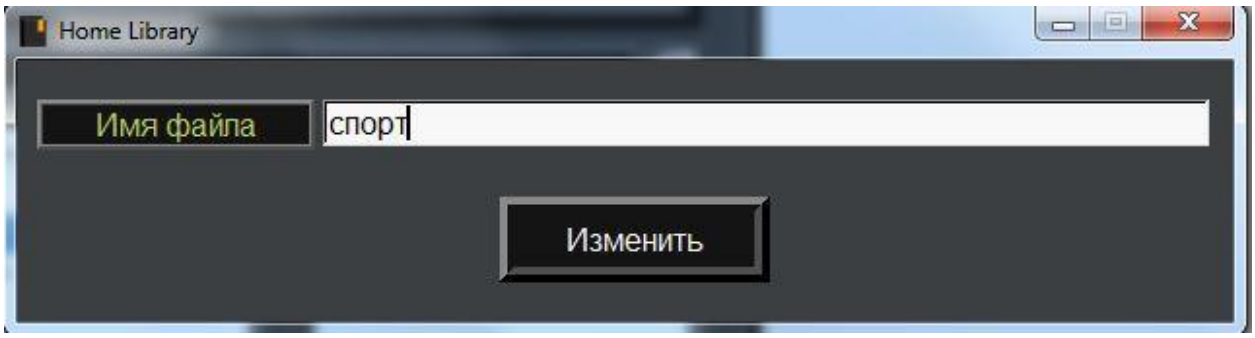


В левом столбце отражается содержимое главной папки, в правом – содержимое выбранной папки. Если в правом столбце выбран файл, то в окне отображается его содержимое. Программа корректно работает с любой структурой папок, но если на одном уровне находятся папки, а на следующем – файлы, программа отобразит файловую структуру наиболее наглядно.

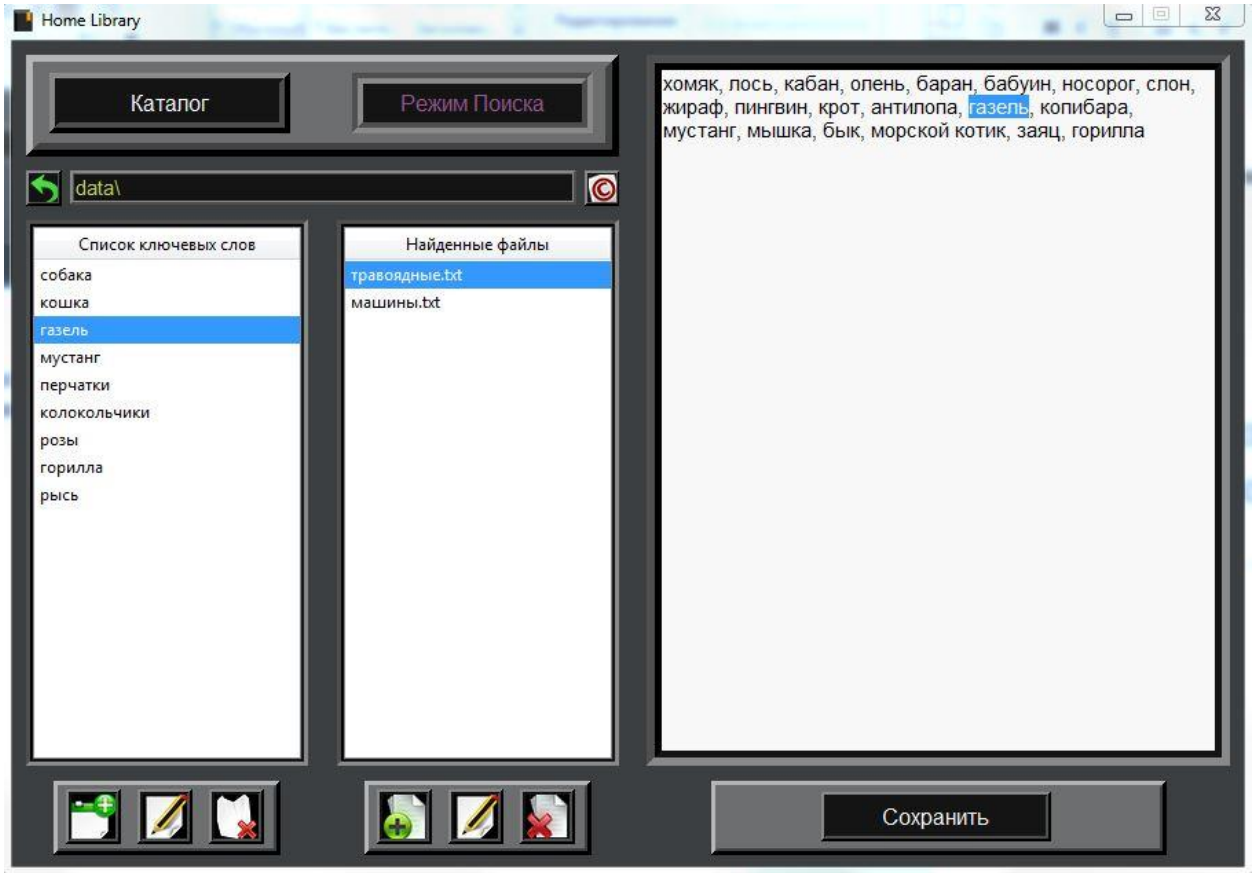
Пользователь может переходить по записям как при помощи компьютерной мыши, так и при помощи клавиш на клавиатуре. С помощью клавиш «Up» и «Down» можно переходить по записям одного столбца, а при помощи клавиш «Left» и «Right» можно переходить от одного столбца к другому. При двойном клике по папке, приложение показывает содержимое данной папки в левом столбце, в случае если в данной папке есть папки нижнего уровня. А при помощи клавиши «Escape» можно перейти обратно к папке верхнего уровня.

При нажатии клавиши «Сохранить», новое содержимое окна отображения сохранится в выбранный в правом столбце файл. Файлы и папки также можно редактировать, изменяя их имена. Для этого были разработаны модальные окна:



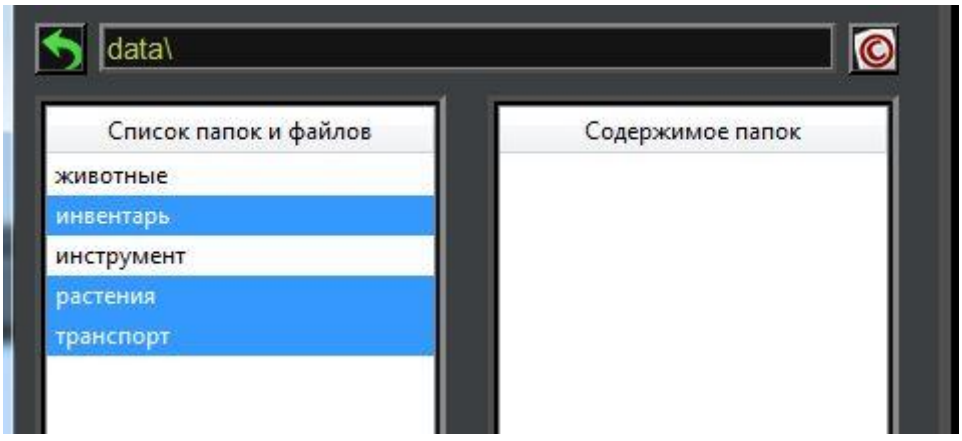


При нажатии на кнопки «Поиск» программа переходит в режим поиска, и окно программы принимает следующий вид:



В левом столбце появляется список ключевых слов, который загружен из файла «KeyList.txt», а в правом – списки файлов, в которых присутствует данное слово. При нажатии на файл он открывается в окне для просмотра файлов, а первое ключевое слово в тексте файла выделяется синим цветом. В режиме поиска также доступно редактирование файлов.

Если искать ключевые слова нужно не во всех папках, то, перед переходом в режим поиска, пользователь может выделить актуальные для него папки. В таком случае поиск ключевых слов будет производиться только в них.



Используемые инструменты и библиотеки

Для разработки данного приложения был выбран язык программирования «python».

Для написания кода использовалась интегрированная среда разработки «JetBrains PyCharm Community Edition».

Для работы с файлами был импортирован модуль «os».

Для возможности быстрого удаления папок со всем содержимым был импортирован модуль «shutil».

Для создания графического интерфейса был импортирован модуль «tkinter».

Для создания столбцов со списками дополнительно был импортирован модуль «ttk», а для отображения окон с предупреждениями модуль «tk_message_box».

Кодировка текстовых файлов

Данное приложение предназначено только для работы с текстовыми файлами, с расширением «*.txt», сохранёнными в кодировке «UTF-8».

Здесь следует отметить важный момент: кодировка «UTF-8» может быть как с «Boom», так и без «Boom».

Отличие заключается только в том что если файл был сохранён в кодировке с «Boom», то в самом начале файла будет записан спецсимвол, указывающий на кодировку файла. При просмотре файла в текстовом редакторе и выводе его содержимого в текстовое поле приложения, данный символ отображаться не будет, но при обращении к нулевому символу содержащегося в файле текста, программа будет получать не первую букву (или цифру), записанную в текстовом файле, а данный спецсимвол, из-за чего могут возникать ошибки и сбои в работе приложения.

Поскольку приложение должно работать с обоими типами файлов, данная проблема была решена при помощи написания нескольких дополнительных строчек кода:

```
if not(text[0].isalnum()):
    word = text[1:]
```

Если нулевой символ не является буквой или цифрой, он удаляется из текста.

Архитектура программы

При написании программы были использованы принципы объектно-ориентированного программирования, то есть вся программа была разбита на небольшие части, которые были прописаны в отдельных классах. Основой программы является класс «MainWindow», в нём были созданы экземпляры остальных классов, таким образом из любого места программы можно обратиться к нужному классу через экземпляр класса «MainWindow».

Все классы были распределены по 5-ти файлам:

- main.py,
- frontend.py,
- backend.py,
- modal_window.py,
- operations.py,
- search.py,
- data.py,
- images.py

Главный файл программы – «main.py», запускает программу:

```
from frontend import *

window = MainWindow()
window.run()
```

В файлах «frontend.py» и «modal_window.py», описан графический интерфейс приложения.

Файл «images.py» нужен для загрузки картинок, используемых для создания кнопок иконки приложения.

В файлах «backend.py» и «operations.py», описана функциональная часть программы.

В файле «search.py» описан механизм поиска.

Файл «data.py», отвечает за работу с файлами, а также создание дерева папок и списка файлов.

Алгоритм создания дерева папок

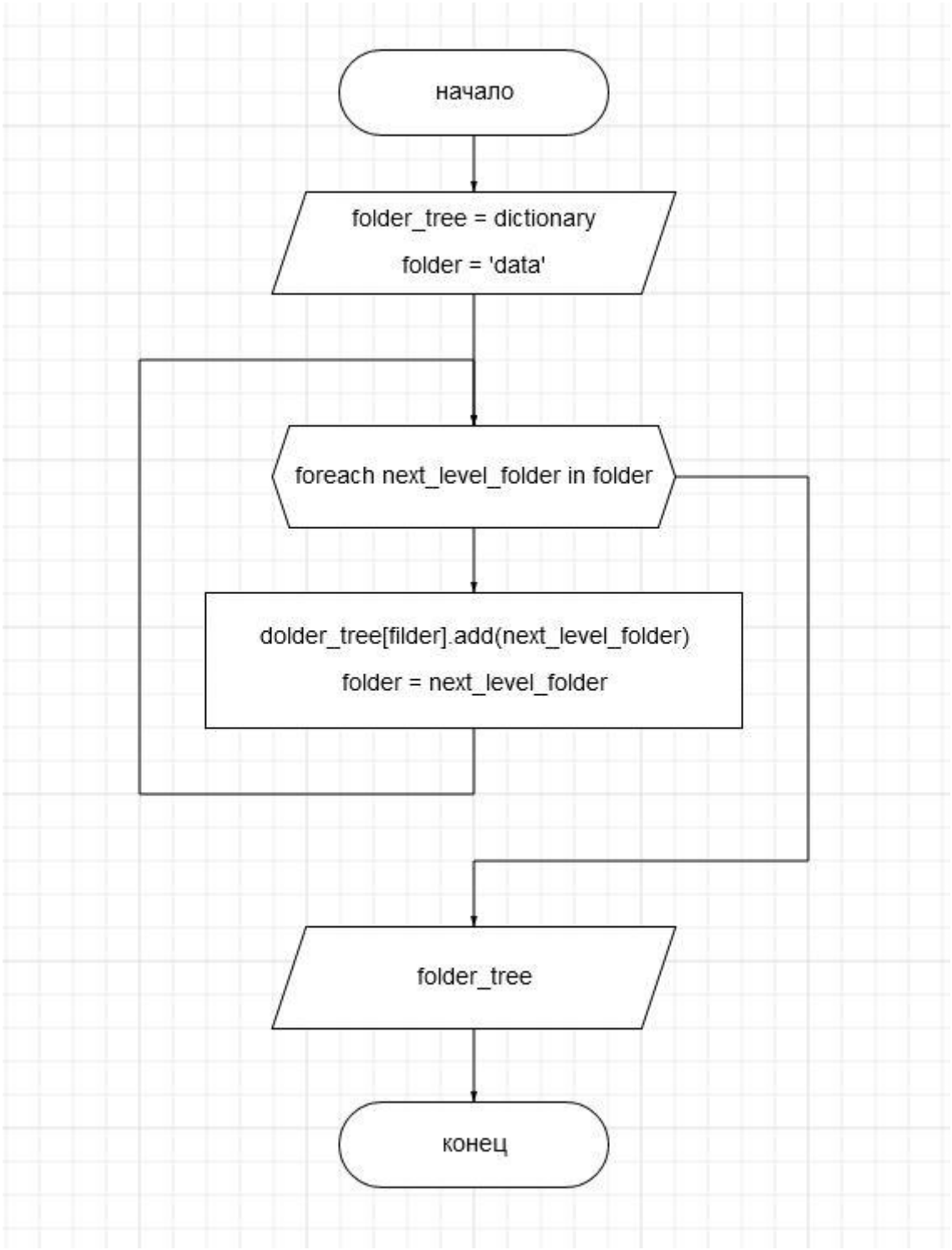
Данное приложение работает со всеми папками, которые находятся в папке «data.txt», которая находится в основной папке проекта. За создание дерева папок отвечает класс «FolderTree».

Для данной цели используется рекурсивный алгоритм обхода графа в глубину. Дерево папок хранится в виде словаря, ключевыми словами которого являются полные адреса всех папок. Значениями данного словаря являются списки, включающие в себя три элемента:

- имя родительской папки
- список папок, содержащихся в данной папке
- список файлов, содержащихся в данной папке

Данные списки включают в себя только папки и файлы первого уровня.

Блок-схема данного алгоритма выглядит так:



А программный код выглядит следующим образом:

```
class FolderTree:
    def __init__(self):
        self.main_folder = 'data'
        self.folder_dict = dict()
        self.file_list = list()
        self.current_folder = self.main_folder
        self.get_folder_dict(self.current_folder)
```

```
    def get_folder_dict(self, object_address, parent_folder_address = ""):
        self.folder_dict[object_address] = [parent_folder_address, list(), list()]
        for next_level_object in os.listdir(object_address):
            next_level_object_address = object_address + "\\" + next_level_object
            next_level_object_address = self.remove_space(next_level_object_address)

            if os.path.isdir(next_level_object_address):
                self.folder_dict[object_address][1].append(next_level_object_address)
                self.get_folder_dict(next_level_object_address, object_address)
            elif next_level_object_address[-4:] == ".txt":
                self.folder_dict[object_address][2].append(next_level_object_address)
                self.file_list.append(next_level_object_address)
```

Параллельно с деревом папок создаётся единый список адресов всех файлов, находящихся в папке «data». Он нужен для более удобного поиска ключевых слов. Если пользователь создаёт поисковый запрос не по всем папкам, а только и избранным, данный список корректируется и лишние файлы из него удаляются.

Переход в режим поиска

Программа может находиться в двух состояниях – в режиме поиска или в режиме каталога. Изначально программа запускается в режиме каталога, но при нажатии кнопки «Поиск» переходит в режим поиска.

Механизм перехода из одного режима в другой прописан в классе «MenuSearshFlag», файла «backend.py». Его смысл заключается в том, что при переходе в режим поиска устанавливается «search_flag», и нужные виджеты главного окна программы собираются заново.

```
class MenuSearchFlag:
    def __init__(self, window):
        self.window = window
        self.search_flag = False

    def change_search_flag(self):
        if self.search_flag:
            self.search_flag = False

            self.window.tree.clear_file_list()
            self.window.tree.create_file_list(self.window.tree.main_folder)

            self.window.menu.menu_buttons()
            self.window.folder_row.hide_search_words_row()
            self.window.folder_row.view_folder_row()
            self.window.file_row.hide_found_files_row()
            self.window.file_row.view_file_row()
            self.window.view.set_file_row()
            self.window.reader_box.clear_text_box()
            self.window.folder_row.folder_row.focus_set()
            self.window.folder_buttons.destroy_word_buttons()
```

```
            self.window.folder_buttons.folder_buttons()
        else:
            self.search_flag = True

            # Если было выделено больше одной папки, создаём массив файлов
            # только по выделенным папкам
            if self.window.tree.current_folder == self.window.tree.main_folder and \
                    len(self.window.folder_row.folder_row.selection()) > 1:
                folder_index_array = self.window.folder_row.folder_row.selection()
                folder_array = list()
                for folder_index in folder_index_array:
                    folder_array.append('data\\' +
                                        self.window.folder_row.folder_row.item(
                                            folder_index)['values'][0])
                self.window.tree.clear_file_list()
                for folder in folder_array:
                    self.window.tree.add_folder_in_file_list(folder)
            # Если было выделено не больше одной папки, создаём массив файлов
            # по всем папкам
            else:
                self.window.tree.clear_file_list()
                self.window.tree.create_file_list(self.window.tree.main_folder)

            self.create_search_settings_main_window()
            if self.window.folder_row.search_words_row.get_children():
                child_id = self.window.folder_row.search_words_row.get_children()[0]
                self.window.folder_row.search_words_row.focus_set()
                self.window.folder_row.search_words_row.focus(child_id)
                self.window.folder_row.search_words_row.selection_set(child_id)
                self.window.view.view_selected_objects_left_search()
            self.window.folder_buttons.destroy_folder_buttons()
            self.window.folder_buttons.word_buttons()

    def create_search_settings_main_window(self):
        self.window.menu.menu_buttons()
        self.window.folder_row.hide_folder_row()
        self.window.folder_row.view_search_words_row()
        self.window.folder_row.insert_search_row()
        self.window.file_row.clear_file_row()
        self.window.file_row.hide_file_row()
        self.window.file_row.view_found_files_row()
        self.window.reader_box.clear_text_box()
```

Алгоритм поиска по ключевым словам

Данная задача реализована достаточно просто, программа по очереди открывает файлы и ищет в них нужные слова. Программный код выглядит следующим образом:

```
class Search:
    def __init__(self, window):
        self.window = window

    def search_files(self, word):
        self.found_file_array = list()

        for file_address in self.window.tree.file_list:
```

```
            file = open(file_address, encoding='utf-8')
            file_text = file.read()
            file.close()
            if file_text:
                if not(word.isalnum()):
                    word = word[1:]
                if not(file_text[0].isalnum()):
                    file_text = file_text[1:]
                if word.strip().lower() in file_text.lower():
                    self.found_file_array.append(file_address)

        self.window.file_row.insert_values_in_found_files_row(self.found_file_array)
```

Отображение файлов с выделенными ключевыми словами

За отображение текста, найденных по ключевым словам файлов, отвечает метод «insert_text_with_key_word», класса «ReaderBoxFrame», находящегося в файле «frontend.py».

Данный метод, помимо текста файла, принимает на вход ещё два параметра: индекс начала выделения и индекс конца выделения и при помощи данных индексов выделяет нужное слово при отображении текста:

```
def insert_text_with_key_word(self, text, begin_index, end_index):
    self.clear_text_box()
    self.text_box.insert(1.0, text.strip())
    self.text_box.tag_add("hg1", '1.' + str(begin_index), '1.' + str(end_index))
    self.text_box.tag_config("hg1", background="#3399FF", foreground="#FFFFFF")
```

Формирует данные индексы метод «view_selected_object_right_search()», класса «ViewOperations», находящегося в файле «backend.py».

Для определения выделенного файла и выделенного слова используются метод «.focus()», «set()» и «selection()» модуля «ttk».

```
def view_selected_objects_right_search(self):
    # Определяем адрес выделенного элемента
    row_id = self.window.file_row.found_files_row.focus()
    if row_id in self.window.file_row.children_dict:
        file_address = self.window.file_row.children_dict[row_id]

        # Определяем выделенное ключевое слово
        key_word = self.window.folder_row.search_words_row.set(
            self.window.folder_row.search_words_row.selection())['word']
        if not(key_word[0].isalnum()):
            key_word = key_word[1:]

        # Открываем файл и извлекаем из него текст
        file = open(file_address, encoding='utf-8')
        file_text = file.read()
```

```
        file.close()

        # Определяем позицию ключевого слова в тексте и создаём нужные индексы
        key_word_index = file_text.find(key_word)
        key_word_end_index = key_word_index + len(key_word)
        if key_word_index == -1:
            key_word_index = 0
            key_word_end_index = 0

        # Вызываем метод, отображающий текст файла и передаём в него нужные индексы
        self.window.reader_box.insert_text_with_key_word(
            file_text, key_word_index, key_word_end_index)
        self.window.operations.displayed_file_address = file_address
```

Горячие клавиши приложения

Для более удобного перехода по файлам и папкам используется ряд горячих клавиш, прописанных в классе «MainWindow», файла «frontend.py».

```
self.root.bind("<Button-1>", self.view.show_selected_objects)
self.root.bind("<Up>", self.view.show_selected_objects)
self.root.bind("<Down>", self.view.show_selected_objects)
self.root.bind("<Right>", self.view.move_focus_right)
self.root.bind("<Left>", self.view.move_focus_left)
self.root.bind("<Double-Button-1>", self.view.go_to_next_folder)
self.root.bind("<Return>", self.view.go_to_next_folder)
self.root.bind("<Escape>", self.view.go_to_prev_folder_event)
```

Функционал данных клавиш прописан в классе «ViewOperations», файла «backend.py». Для примера рассмотрим метод «move_focus_left()».

```
def move_focus_left(self, event):
    # Убеждаемся в том, что фокус находится в правом поле (режим каталога)
    if len(str(self.window.root.focus_get())) == 35:
        if int(str(self.window.root.focus_get())[24]) == 4:

            # Если в левой части уже есть выделенный элемент ставим курсор на него
            if self.window.folder_row.folder_row.selection():
                child_id = self.window.folder_row.folder_row.selection()
            # Если в левой части нет выделенных элементов,
            # ставим курсор на первый по списку
            else:
                child_id = self.window.folder_row.folder_row.get_children()[0]
            self.window.folder_row.folder_row.focus_set()
            self.window.folder_row.folder_row.focus(child_id)
            self.window.folder_row.folder_row.selection_set(child_id)

            # Удаляем текст файла, который был выделен из поля для просмотра файла
            self.clear_file_text_frame()
            self.set_file_row()

    # Убеждаемся в том, что фокус находится в правом поле (режим поиска)
    if len(str(self.window.root.focus_get())) == 36:
```

```
        if int(str(self.window.root.focus_get())[24]) == 4:

            # Если в левой части уже есть выделенный элемент ставим курсор на него
            if self.window.folder_row.search_words_row.selection():
                child_id = self.window.folder_row.search_words_row.selection()
            # Если в левой части нет выделенных элементов,
            # ставим курсор на первый по списку
            else:
                child_id = self.window.folder_row.search_words_row.get_children()[0]
            self.window.folder_row.search_words_row.focus_set()
            self.window.folder_row.search_words_row.focus(child_id)
            self.window.folder_row.search_words_row.selection_set(child_id)
            self.clear_file_text_frame()

            # Удаляем текст файла, который был выделен из поля для просмотра файла
            self.window.file_row.clear_found_files_row()

            # Узнаём новое выделенное ключевое слово и создаём для него список файлов
            key_word = self.window.folder_row.search_words_row.set(child_id)['word']
            self.window.search.search_files(key_word)
            self.window.folder_row.search_words_row.selection_set(child_id)
            self.clear_file_text_frame()

            # Удаляем текст файла, который был выделен из поля для просмотра файла
            self.window.file_row.clear_found_files_row()

            # Узнаём новое выделенное ключевое слово и создаём для него список файлов
            key_word = self.window.folder_row.search_words_row.set(child_id)['word']
            self.window.search.search_files(key_word)
```

Аналогичным образом работают и остальные методы горячих клавиш.

Заключение

Было разработано приложение «Home Library», которое позволяет удобно и быстро просматривать папки и содержащиеся в них файлы вместе с их содержимым. В режиме поиска данное приложение позволяет быстро находить нужные файлы по ключевым словам. При необходимости пользователь может редактировать список ключевых слов и содержимое файлов через интерфейс приложения.

Данное приложение может быть полезно для хранения какой-либо систематизированной информации, например личного справочника программиста, который он может пополнять по мере приобретения новых знаний, а при необходимости быстро находить в нём нужную информацию.

С практической точки зрения разработка данного приложения дала представление о работе с файловой системой и новый опыт в разработке графических приложений.