САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

# Алгоритмы и структуры данных

*1.2.1 Перекрестные ссылки. Бинарное дерево поиска*

«[*Описание спецификации программы*»](https://dl.spbstu.ru/mod/assign/view.php?id=93408)

Студент гр. 3530904/90002

Ф.И.О Мэн Цзянин

1. **Общая постановка задачи**
   1. Для разрабатываемого словаря реализовать основные операции:
      * INSERT (ключ, значение) – добавить запись с указанным ключом и значением
      * SEARCH (ключ)- найти запись с указанным ключом
      * DELETE (ключ)- удалить запись с указанным ключом
   2. Предусмотреть обработку и инициализацию исключительных ситуаций, связанных, например, с проверкой значения полей перед инициализацией и присваиванием.
   3. Программа должна быть написана в соответствии со стандартом программирования: C++

Programming Style Guidelines (<http://geosoft.no/development/cppstyle.html>).

* 1. Тесты должны учитывать как допустимые, так и не допустимые последовательности входных данных.

1. **Требования**

Разработать и реализовать алгоритм формирования перекрестных ссылок:

* прочитать текст и вывести его с добавлением последовательных номеров строк;
* собрать все слова, встречающиеся в тексте;
* сформировать таблицу, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список строк его нахождения (по возрастанию номеров строк)

Для реализации задания использовать бинарное дерево поиска, узел которого может содержать:

* Ключ – слово
* Информационная часть – ссылка на список, содержащий номера строк

1. **Спецификация**
   1. Файл должен быть открыт
      1. В противном случае выводим ошибку, содержащую текст: «cant\_open\_the\_file»
   2. root, который в бинарном дереве поиска нельзя было nullptr
   3. все данные должны иметь правильный тип
      1. Поля Номер строки(row), столбца(col), счетчик(count) и длина слова(length) должны иметь тип данных unsigned int
      2. Номер строки, столбца и длина слова должны больше чем 1
   4. Если пустая строка пропускается, счетчик не накапливается
   5. Для повторяющих слов
      1. В одной статье может быть несколько повторяющихся слов,и номера строк и столбцов этих слов должны храниться в определенном контейнере
      2. При печати функции(printWordAndInfo), которая выводит слова, должен быть предоставлен внешний интерфейс, чтобы пользователь мог выбрать, следует ли печатать повторяющиеся слова
         1. Если «bool repetitive\_word = true»: выводит все повторяющиеся слова вместе с их информацией
         2. Если «bool repetitive\_word = false»: выводит это слово, номер строк и столбцов, которые появляются в первый раз
   6. Для одного и того же слова, но с большой буквы или с “ ’s ”. Все буквы должны быть преобразованы в маленькую букву.

**Пример:**

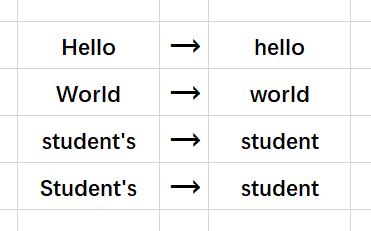


Рис.1

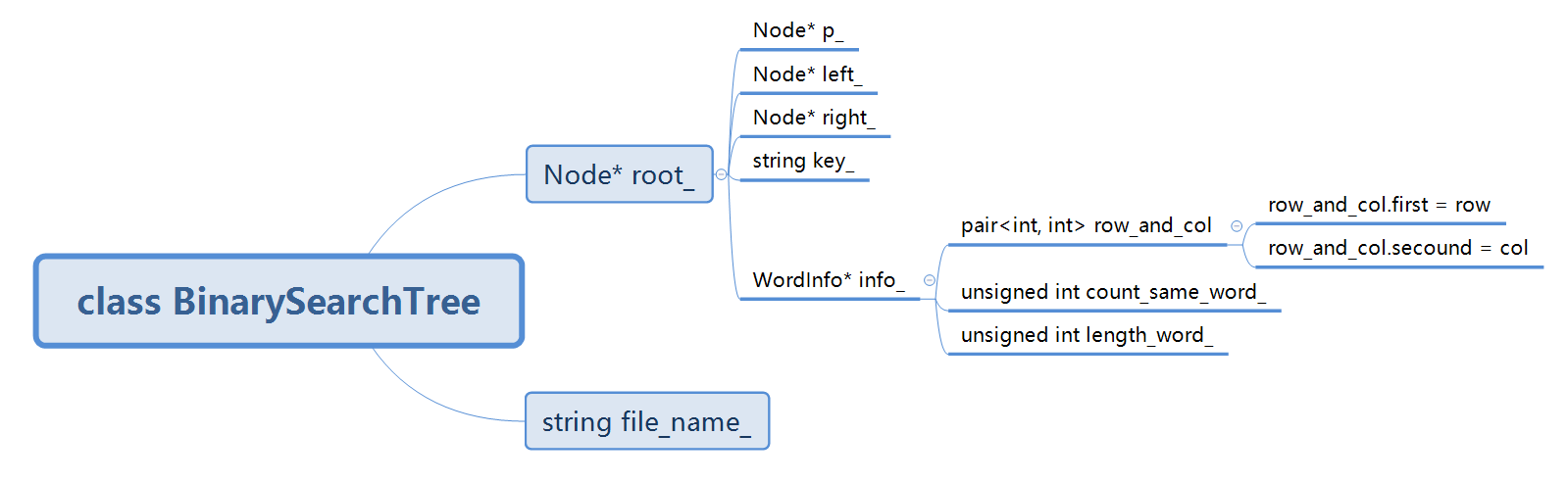
1. **Структура данных**

Рис.2

1. **Интерфейс**

* Сохранить имя файла, которое пользователь вводит

void openFile(string file\_name);

* Выводит текст и вывести его с добавлением последовательных номеров строк в файл

void printTextWithRowNum();

* Найти запись с указанным ключом

void searchWord(string word\_search);

* Удалить запись с указанным ключом

void deleteWord(string dele\_word);

* Собрать все слова, встречающиеся в тексте

void makeTree();

* Печать на экране

void outputDictionaryOrderInFile(bool repetitive\_word);

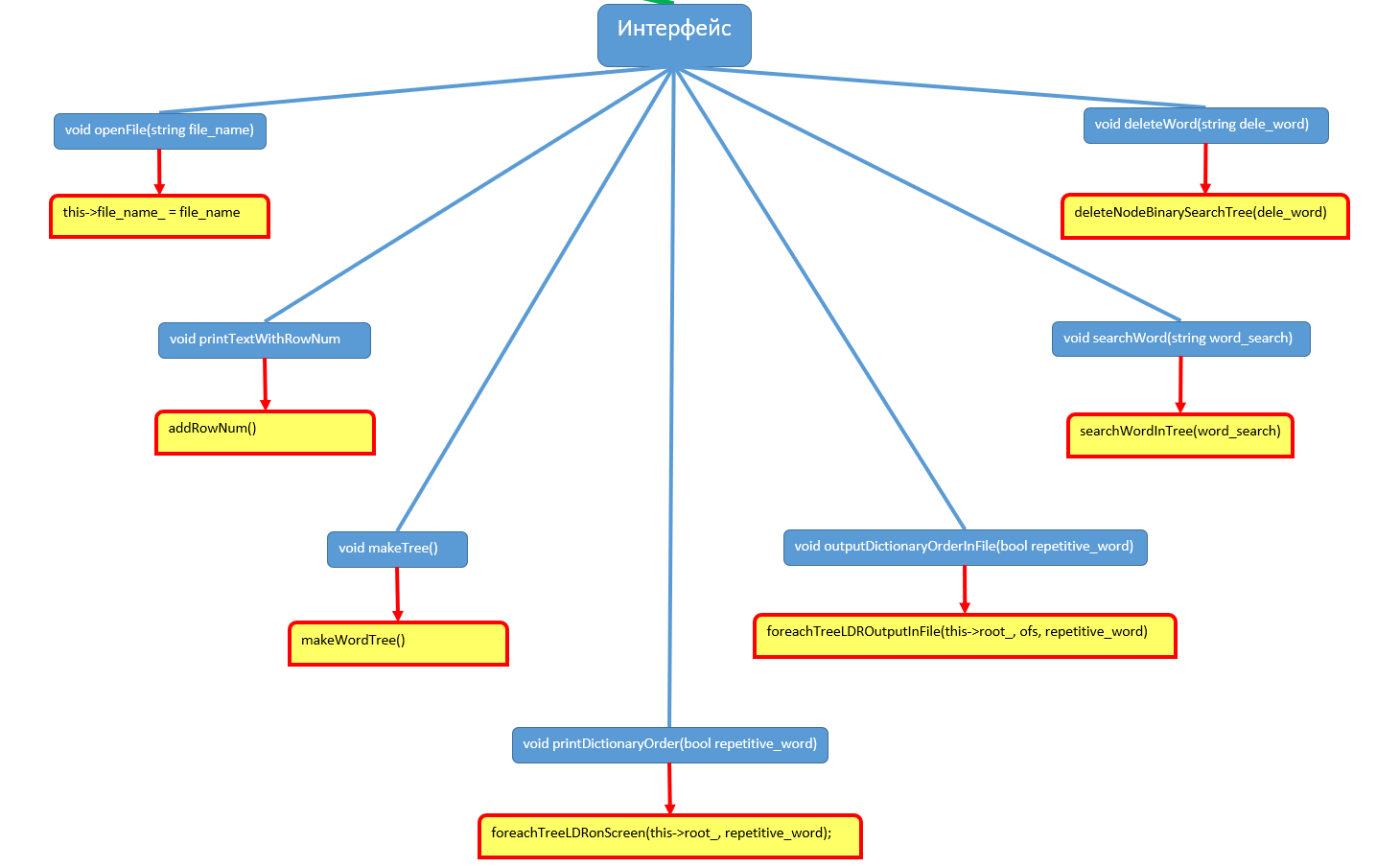
* true == repetitive\_word

вывод повторяющихся слов

* false == repetitive\_word

не вывод повторяющихся слов

* Выводить в фале

void outputInFile();

1. **Методы**

* **void insertNode(Node\* nodeInsert, unsigned int count\_word\_text);**

Каждое слово и его информации (row, col, count, length) – это один узел в бинарном дереве поиска. Функция этой функции заключается в вставке узлов в двоичное дерево поиска.

* **void makeWordTree();**

С помощью предыдущей функции (void insertNode(Node\* nodeInsert, unsigned int count\_word\_text);), эта функция вставляет все слова из статьи в двоичное дерево поиска по лексикографическому порядку, После преобразования слова в нижний регистр и удалил “.”, “.”, “’s”.

Пример:

*“This Prestwick House Literary Touchstone Edition includes a glossary*

*and reader's notes to help the modern reader fully appreciate London's masterful weaving of science, philosophy, and the storyteller's”*

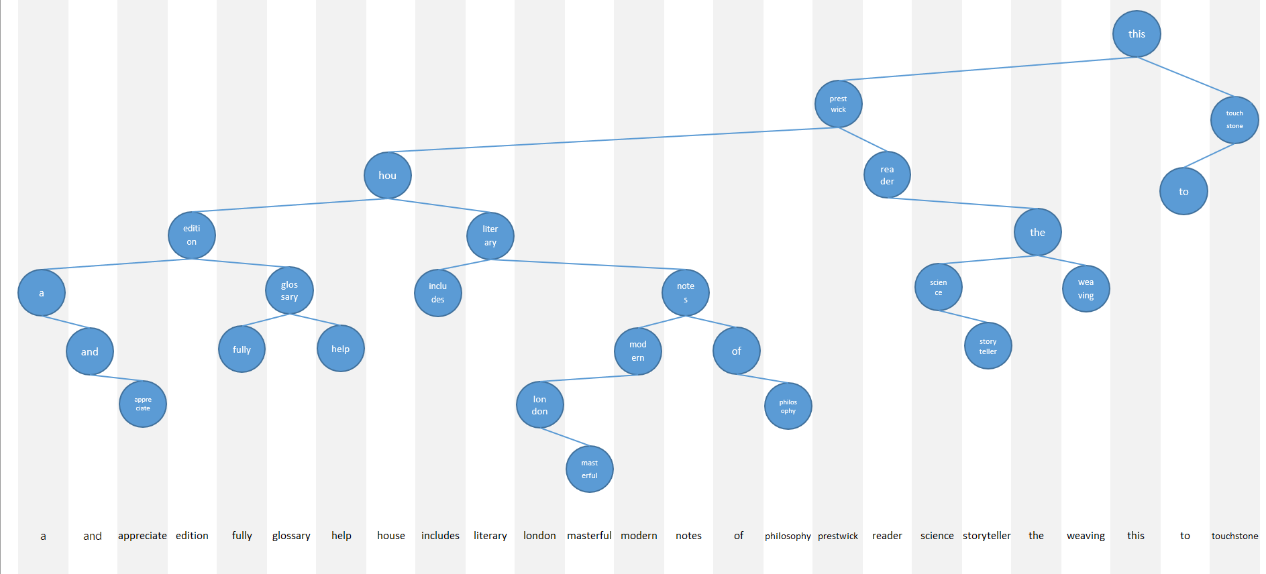
Бинарное дерево поиска должно:

Рис. 3 бинарное дерево поиска

* **void addRowNum();**

Эта функция считывает статью из файла и выводит статью с добавленным номером строки в output.txt. Кроме того, она также подсчитывает общее количество слов в статье, количество строк.

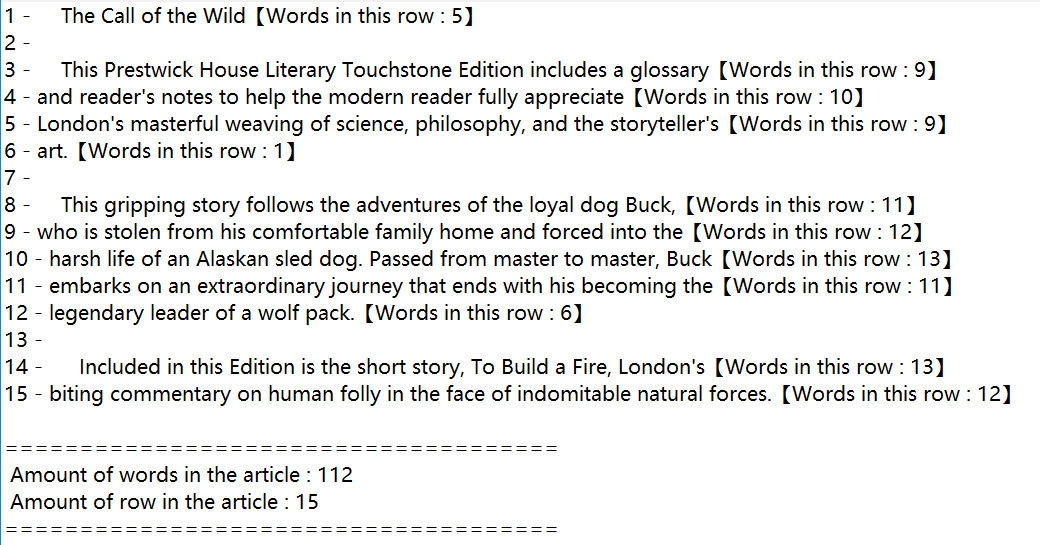


Рис.4 метод void addRowNum()

* **void foreachTreeLDRonScreen(Node\* root, bool repetitive\_word);**

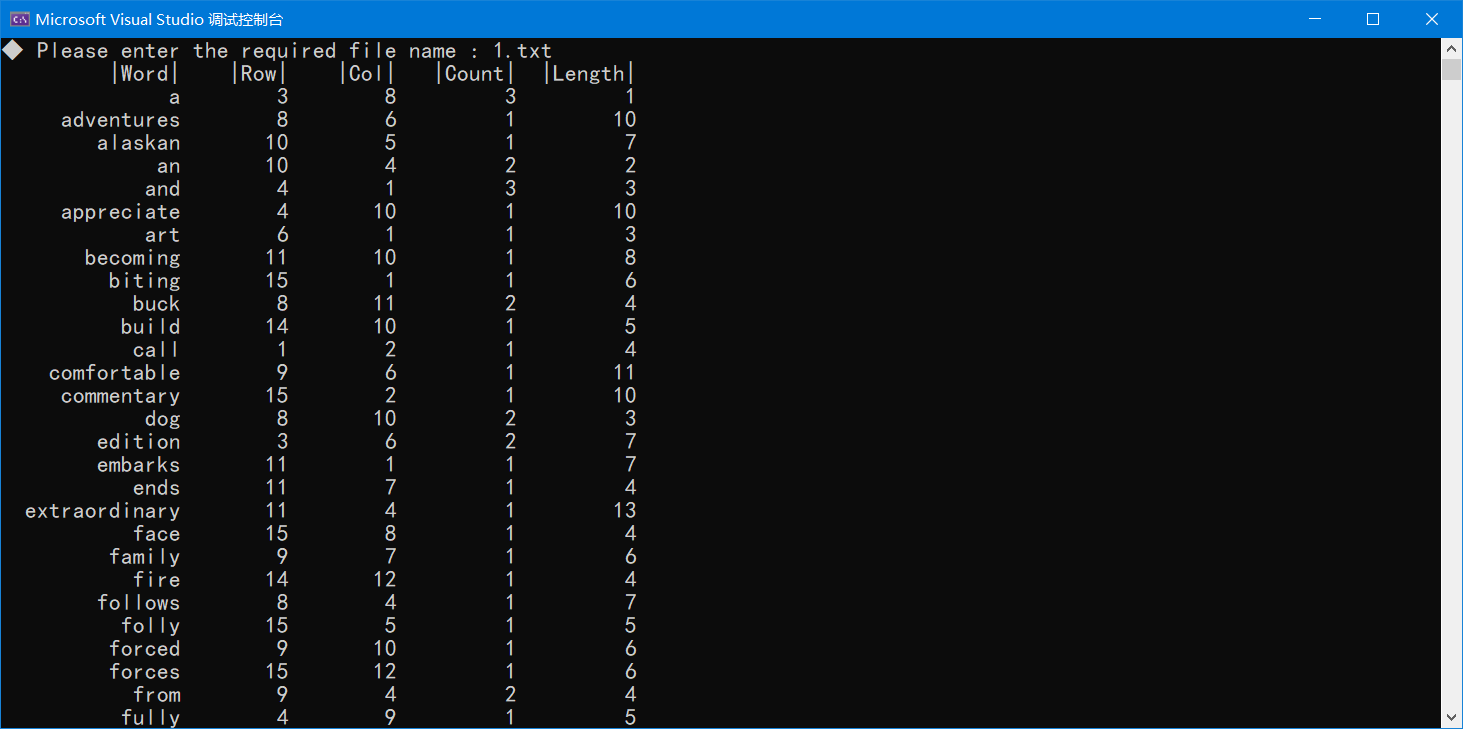
Сформировать таблицу и печать на экране, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список строк, столбец его нахождения (по возрастанию номеров строк), количество одинаковых слов и количество букв в слове. В аргументе функции **bool repetitive\_word** – это выводит ли повторные слова.

Рис.5 Сформировать таблицу и печать на экране

* **void foreachTreeLDROutputInFile(Node\* root, ofstream& ofs, bool repetitive\_word);**

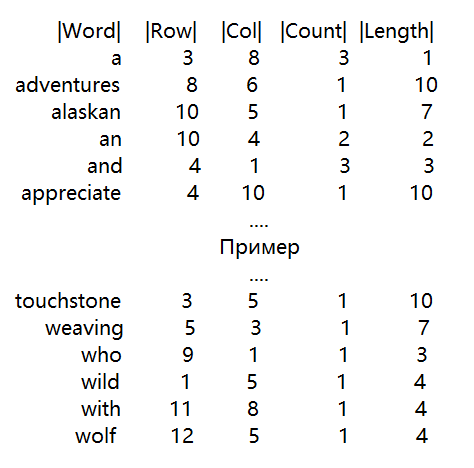
****Сформировать таблицу и выводить в фале, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список строк, столбец его нахождения (по возрастанию номеров строк), количество одинаковых слов и количество букв в слове. В аргументе функции **bool repetitive\_word** – это выводит ли повторные слова

Рис.6

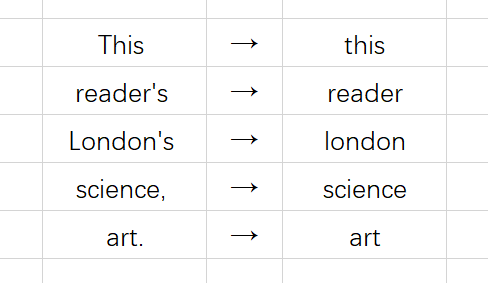
* **void changeCaseAndClearSign(string& word);**

Рис. 7

Для одного и того же слова, но с большой буквы или с “ ’s ” или на конце слова имеет “ **,**”, “ **.**”. Эта функция может преобразовать все буквы в маленькую букву и удалить “ **,**”, “ **.**”.

* **bool isHaveSameWord(string word, pair<unsigned int, unsigned int> row\_col\_word)**

Эта функция предназначена для определения того, существует ли одно и то же слово в двоичном дереве поиска при вставке нового слова. Если есть однородное слово **return true**, если нет **return false.**

* **Node\* searchWordInTree(string word\_search);**

Поиск, существует ли введенное пользователем слово.

Пример:

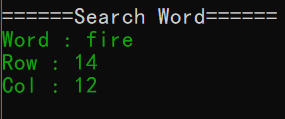
* Искать fire

Рис.8 искать fire

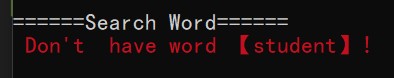
* Искать student

Рис.9 искать student

* **string& clearAllSpace(string& str);**

Чтобы удаление пробелов в строке, при вычислении длины строки.

* **bool deleteNodeBinarySearchTree(string dele\_key);**

Удаляет узлы двоичного дерева поиска на основе введенных пользователем слов.

Пример:

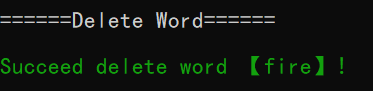
* удалить file

Рис.10 удалить file

* удалить student

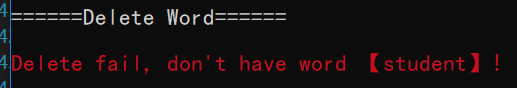


Рис. 11 удалить students

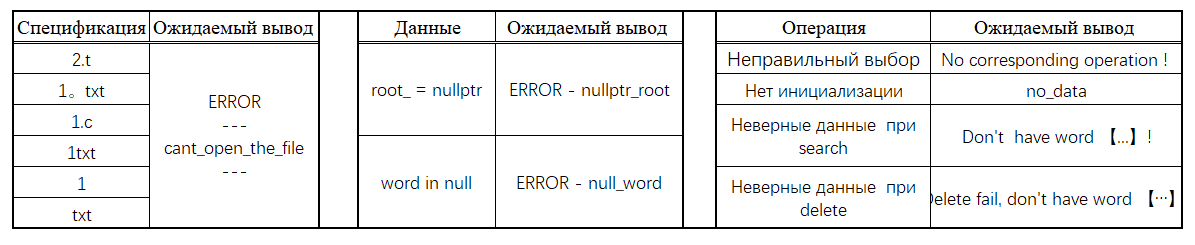
1. **Для некорректного ввода данных:**

Таблица 1