**Принцип работы  
text.AI**

**#define DEPTH 4** – количество слов, которое берётся для выборки.

**1)** Объявляем 2 строки на исходный и проверяемый тексты.

**2)** Печатаем интро с помощью **printIntro()**.

**3)** Получаем каждый текст с помощью **getline()** .

**4)** Каждый текст передаём в функцию **textConvertation()**, которая возвращает уже отформатированный текст.  
  
Принцип работы алгоритма в **textConvertation()**:   
 - Объявляем **temp** строку для хранения отформатированного текста  
 - Создаём цикл, который будет проверять каждый элемент полученного в параметры текста   
 - Если элемент равен ‘ ‘ – прибавляем к **temp**   
 - Если элемент равен по ASCII [65 - 90] (Большие буквы),то прибавляем 32 (Расстояние между большой и маленькой буквами) и прибавляем к **temp**  
 - Если элемент равен по ASCII [48 – 57] (Цифры) или [97 - 122] (Маленькие буквы) - прибавляем к **temp  
  
4)** Объявляем 2 int переменные для подсчёта количества слов в тексте и инициализируем их с помощью функции **howManyWordsTextHas()**.   
  
Принцип работы алгоритма в **howManyWordsTextHas()**:   
 - Объявляем int переменную **counter** для подсчёта пробелов в тексте и присваиваем ей значение 1 (Так как разность в количестве между пробелами и словами равна 1(1 пробел = 2 слова и т.д.))

- Проверка если пользователь ввёл строку, которая меньше, чем **DEPTH**Возвращаем полученное число.

**5)** Инициализируем 2 массива строк с полученными размерностями (из пункта **4**).

**6)** Заполняем 2 массива строк с помощью **fillingArray()**.  
  
 Принцип работы алгоритма в **fillingArray()**:

- Объявляем int переменную **wordsCounter** для подсчёта количества слов  
- Объявляем string **word** для хранения слова  
- Создаём цикл, который будет проверять каждый элемент полученного текста переданного в параметры  
- Если элемент равен ‘ ‘ или концу строки (для того чтобы «захватить» последнее слово) –   
добавляем **word** в массив, который был получен в параметры, с индексом **wordsCounter**. Инкрементируем **wordsCounter** на 1 и обнуляем **word**.  
- Если элемент не равен пробелу или концу строки, то добавляем элемент в **word**

**7)** Объявляем переменную **percentageOfPlagiarism** и инициализируем её с помощью функции **calculatingPercentageOfPlagiarism()**.  
  
Принцип работы алгоритма в **calculatingPercentageOfPlagiarism()**:  
 - Объявляем 2 переменные для подсчёта выборок(**allSelections** для всех, **plagiatedSelections** для выборок, который алгоритм определит как плагиат  
 - Объявляем 2 переменные для подсчёта «шагов» в каждом тексте(Под «шагом» в данном случае нужно понимать взятие слов для выборки(1, 2, 3; 2, 3, 4 и т.д.))  
 - Объявляем 2 **temp** строки для временного хранения выборок **Алгоритм поиска:** - Сначала первый цикл для «ходьбы» по проверяемому тексту  
 - Второй цикл для «ходьбы» по исходному тексту  
  
 Конструкция ((sizeOfText - DEPTH) + 1) – математическое вычисление при котором подсчитывается количество возможных выборок при разной **DEPTH**.   
 Например:  
 - Если **DEPTH** = 4, а количество слов в тексте 20,то:  
 20 – 4 + 1 = 17, т.е для выборки это 17,18,19,20 слова- Если **DEPTH** = 3, а количество слов в тексте 20,то:  
 20 – 3 + 1 = 18, т.е для выборки это 18,19,20 слова

- Третий цикл нужен для того, чтобы заносить в **temp** количество слов до **DEPTH.**  
Конструкция **k + step** нужна чтобы переходить к следующим элементам тексте на шаг вперёд (Позже **step** будет инкрементироваться по 1)  
  
- Если 2 **temp** строки оказались равны – увеличиваем **plagiatedSelections,** обнуляем **temp,** выходим из цикла  
- Если нет, то увеличиваем «шаг», обнуляем **temp**- После полного прохода по исходному тексту увеличиваем **allSelections** и **step** для проверяемого тексте. **Step** для исходного обнуляем, чтобы проверка происходила сначала.  
  
  
- Возвращаем в процентах отношение «сплагиаченных» выборок ко всем.

**8)** С помощью **printPercentage**() выводим на экран процент плагиата.

**9)** Завершаем программу.