**JAGAN INSTITUTE OF MANAGEMENT STUDIES**   
3, institutional area, sector-5, Rohini, Delhi-110085   
(AFFILIATED TO GGSIPU-NEW DELHI) 

**Object Oriented Programming with C++**  
**(BCA 205)**

Assignment-2

Lexical Analysis – File Handling in C++

**Submitted by:**

Manawi Choudhary                                                          Himanshi Tomer   
Enrolment no.- 04314002023                          Enrolment no.- 05214002023  
BCA, Shift-1, Semester-3 BCA, Shift-1, Semester-3

# Table of Contents

|  |  |  |
| --- | --- | --- |
|  | **Content** | **Page No.** |
| 1. | Introduction | 2 |
| 2. | Case Description | 3 |
| 3. | Contributions of Team Members | 5 |
| 4. | Objectives | 6 |
| 5. | Data Description | 7 |
| 6. | File Descriptions | 9 |
| 7. | Source Code | - |
| 8. | Screenshots | 14 |
| 9. | Conclusion | 23 |

# Introduction

Lexical Analysis, also known as scanning, is the process of reading the source code or a dataset, character by character and segregating or grouping them under certain conditions. The concept is considered a fundamental step in NLP (Natural Language Processing), which is a part of Machine Learning that gives devices the ability to interpret, manipulate and understand human language.

Lexical Analyzer

Read characters

Token

Push Back Extra Char

Ask for Token

Lexical Analysis uses tokens (sequence of characters) that are then differentiated and interpreted.

e.g.  
 Sample Data:  
 *This is another assignment for OOPS since first one got branded ChatGPT. This was   
 coded, compiled and documented by Manawi and Himanshi from Shift-1 for batch of   
 2023-26.*

In the given data, the token so formed are:  
 *'This', 'is', 'another', 'assignment', 'for', 'OOPS', 'since', 'first', 'one', 'got', 'branded',   
 'ChatGPT', '.', 'This', 'was', 'coded', ',', 'compiled', 'and', 'documented', 'by', 'Manawi',   
 'and', 'Himanshi', 'from', 'Shift-1', 'for', 'batch', 'of', '2023-26', '.'*

With lexical analysis, file handling is incredibly essential for the management of the data that we use, especially in C++. For lexical analysis and the result so obtained, all are stored in files which would be inaccessible if it weren’t for file handling operations.

C++ provides File Handling classes, ifstream, ofstream, and fstream for all tasks files. It allows storage, retrieval and modification for input and output data results. It is also far more ideal for large dataset processing and retrieval as and when required.

This assignment focuses on implementing lexical analysis using a C++ program that reads, splits, processed and divides the data. Through file handling, it executes how the data is handled.

# Case Description

**Problem Description:**

You are given a dataset of a specific datatype in a text file. Design a program in C++ which reads all data and performs a lexical analysis of each individual data unit (integers, characters, words etc). Now create separate text files for each lexical unit. Name all the files as the name of each unit respectively. Every file must have first value as the total count of that unit stored in that file. Make sure that dataset contains at least 1000 units of data and should be of same type. The program should work even if another dataset is taken as input of some other datatype.

Example: a given file “sample.txt” contains a research article of appx 5000 words   
Output :

1. Words.txt: contains all the words from source file

2. Numbers.txt: contains all numeric data from source file

3. Mixed.txt: contains all words that contain numeric digits e.g. 5A, 64E, word2vec etc

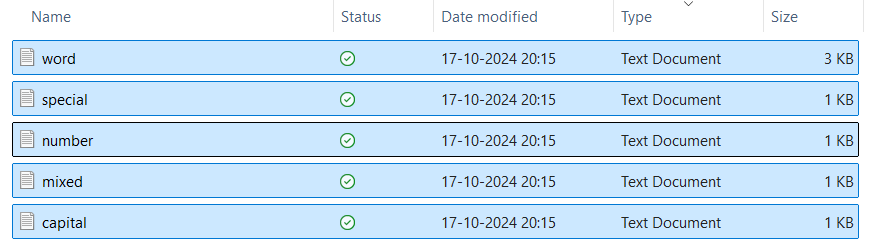
4. Special.txt: contains all special symbols used in the source file

5. capitalWords.txt: contains words that were all capitalized like INDIA, IIT

**Proposed Solution:**

For the required lexical analysis, file handling and multiple loop constructs and conditional statements have been used.

The source code is written such that it takes a text file (dataset), reads each element, whether alphabet, number, special character, spaces, newlines, etc. Once read, it analyses what type of data it is and splits the contents in various different files which are created upon executing the executable (.exe) file.



The text is grabbed as words (let’s call it tokens since that’s the proper terminology). The token so grabbed undergoes conditional statements character by character to be assessed itno it’s type and cleaned if the requirement be so.   
For example,  
 Data: *word2sec, INDIA, words, mi&ed, etc,…* Tokens: ‘*word2sec’, ‘INDIA’, ‘words’, ‘mi&ed’,…* Word.txt: word2sec  
 INDIA  
 words  
 mi&ed

As it does so,… dedicated counter variables are set, priorly initialized to zero to count how many

The following are the steps involved in the successfully implementation:

1. Including
2. Initialization of files using command line arguments such that the program isn’t limited to one particular dataset.

The given is a simple, Adjacency Matrix implementation of the algorithm.

# Contributions of Team Members

The following is the broader division of work performed by the team member:

Manawi Choudhary  
(04314002023)

* + Coding:
    1. Data File Handling Implementation
    2. Looping Constructs
    3. Boolean Flag Variables
    4. Counter Variables
    5. Conditional Statements
    6. Position Marker Pointers

Himanshi Tomer  
(05214002023)

* + Coding:
    1. Conversion to Command Line Arguments Compatibility
    2. Cleaning of words for writing
    3. String Termination
    4. Boolean Flag Variables
    5. Conditional Statements
    6. Variable Initialization

Additional tasks performed together:

* + Formatting of Code
  + Commenting
  + Documentation (Written and Edited)
  + Checking source against multiple data sets.

# Objectives

The objective of this assignment is to:

1. Executing a Lexical Analysis:   
   The program performs a lexical analysis on a dataset, text type file taken in via command line arguments. It reads the data and categorizes them on the basis of whether they’re words, numbers, special characters, capital words, mixed (words with numerals), etc.
2. Categorizing Tokens:   
   To implement classification of token (sequence of data, aka words) by proper identification, and storing them in only the right place for them. That each token is taken as a unit, read character by character and none are skipped or ignored.
3. Implementing File Handling: File Handling using C++, gives the project a more generalized edge that makes it usable for all types of datasets rather than being constrained to one. It also allows to store results and not disturb the existing dataset. Unlike the terminal, large data processing is also more convenient.
4. Using Command Line Arguments:  
   To allow the program to be more accessible, flexible and user friendly, command line arguments offer the required functionality. The program no longer dependent on the source code inputs.

# Data Description

Data Structures

* **File Management:**
  + Attributes:
    - Input Objects
      * *fstream fin* (for reading data from file)
    - Output Objects
      * *ofstream special\_file*
      * *ofstream word\_file*
      * *ofstream capital\_file*
      * *ofstream number\_file*
      * *ofstream mixed\_file*
  + Operations:
    - Open and close files for reading and writing
    - Read data from object *fin* and write to files using various *ofstream* objects
* **Counter Variables:**
  + Attributes:
    - Integer type data variables
      * *special\_count*
      * *word\_count*
      * *capital\_count*
      * *number\_count*
      * *mixed\_count*
  + Operations:
    - Increments each time something is written to the respective files they were created for.
* **Variables:**
  + Reading:
    - To read individual characters
      * *char ch*
    - To read whole words
      * *char str [1024]*
  + Boolean Flags:
    - *special*
    - *capital*
    - *capital\_flag*
    - *number*
    - *mixed\_alpha*
    - *mixed\_num*
    - *mixed\_special*
  + Cleaning:
    - *char clean\_n [1024]* with *int ind\_n*
    - *char clean\_cap [1024]* with *int ind\_cap*
    - *char clean\_spc [1024]* with *int ind\_spc*

Algorithms

* Data structures:   
   Use of character arrays for storing input strings and cleaned versions of them.
* Implementation:   
   Read from files, traverse characters, classify into categories. Clean if required.
* Steps:
  1. Initialize counter variables.
  2. Read each word from dataset.
  3. Traverse the word char by char.
  4. Classify the word or character into respective category.
  5. Write data as categorized to respective files .
  6. Write the total count to output files.

Design Decisions

* **Input Output handling:**
  + Command Line Arguments used for more user friendly and flexible solution.
* **Token Representation:**
  + Character Array used to store words and their clean versions to allow analysis character by character
* **Lexical Analysis:**
  + Boolean flags used to check during each traversal.
* **1024 as Size:**
  + Takes into consideration the maximum length possible for human language words, while also accounting for large data set numerical data, etc.
  + Fixed-size makes memory management more effective.

Complexities:

* **Time complexity:**   
  Given lexical analysis has a time complexity of O(n), where n is the number of characters in the dataset file since each character is to be processed for analysis.
* **Space complexity:**   
  The code requires O(k) space, where k represents maximum length of the word. Which here would be 1024.

# File Descriptions

The following are the files that have been used to successfully compile the assignment and get the desired output:

1. Program File - *final\_console\_version.cpp*
2. Executable File - *a.exe*
3. Data File - *sample.txt*

The format of the files is that of a text file and a c++ file with the respective extensions.

While *final\_console\_version.cpp* contains our code, on compilation in cmd, it creates a *.exe* file of the same name. The *.exe* file represents the executable file of the code so written.

The *.txt* holds mixed datatype data that are taken as sample data sets to run though the assignment code, undergo lexical analysis and check whether the output delivers desired results.

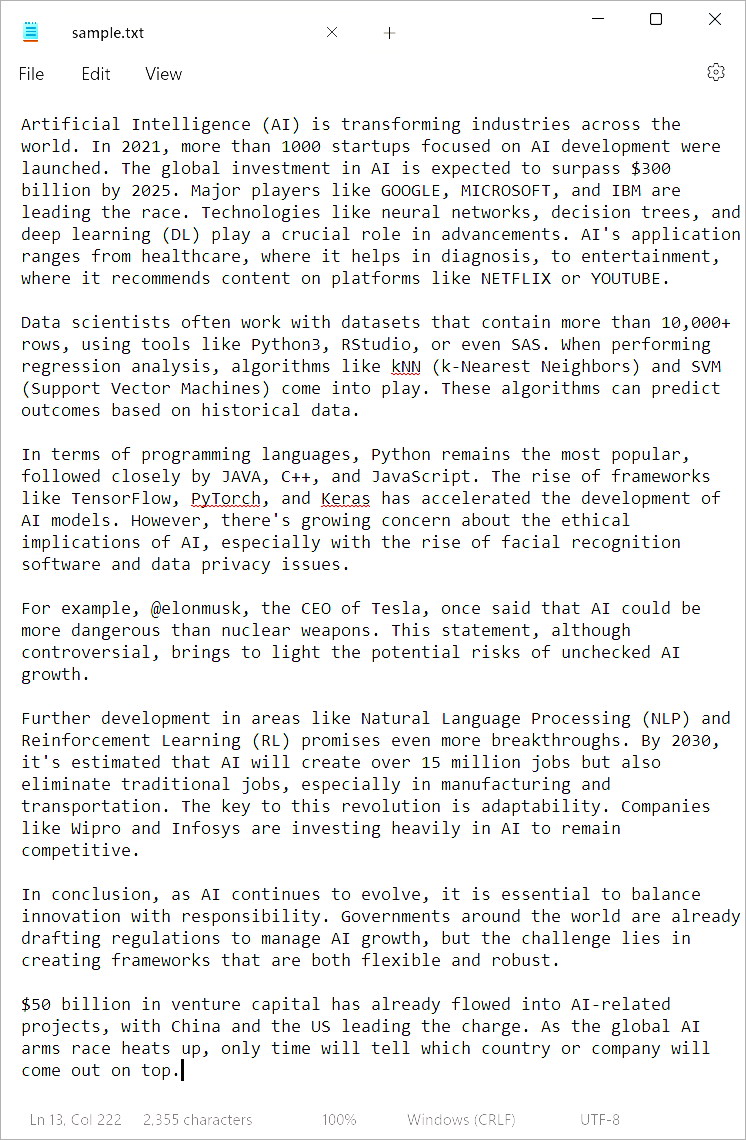
Description of Files:

1. **final\_console\_version.cpp File:**
   1. This file contains the C++ code for the lexical analysis.
   2. Includes all required header files such as *iostream, fstream, cstring,..* and the main() function that opens, closes files and works on them.
   3. Uses a g++ compiler to execute.
   4. Format:   
       .cpp (C++ source code file)
   5. Purpose:   
       This file implements the solution to the problem, opens files, read them, writes to them, closes them, processes the data read, segregates the data so read and cleans it up.
2. **a.exe File:**
   1. This is the executable file generated after compiling the.cpp file. It can be directly run to execute the program without needing to recompile the source code.
   2. Since we follow the command line arguments convention, the exe file is called in cmd with it’s required arguments.
   3. Format:   
       .exe (Windows executable file)
   4. Purpose:   
       This file represents the compiled version of the program, which can be executed to solve the problem without re-running the compilation process.
3. **sample.txt:**
   1. Contains the sample data for the program testing.
   2. It stores the sample data sets that the program reads and processes.
   3. Format:   
       .txt (text file)
4. **Output Files**:
   1. Five output files are created per the names given to call the .exe in cmd.
   2. It stores the analysed content for each of the characters from the dataset, depending on where it belonged.
   3. Format:   
       .txt (text file)
   4. The first line of each .txt contains the following statement:  
       *Total numbers of \_\_\_\_\_\_\_\_ characters:- \_\_*This holds the type of characters name and number of characters in the file created.

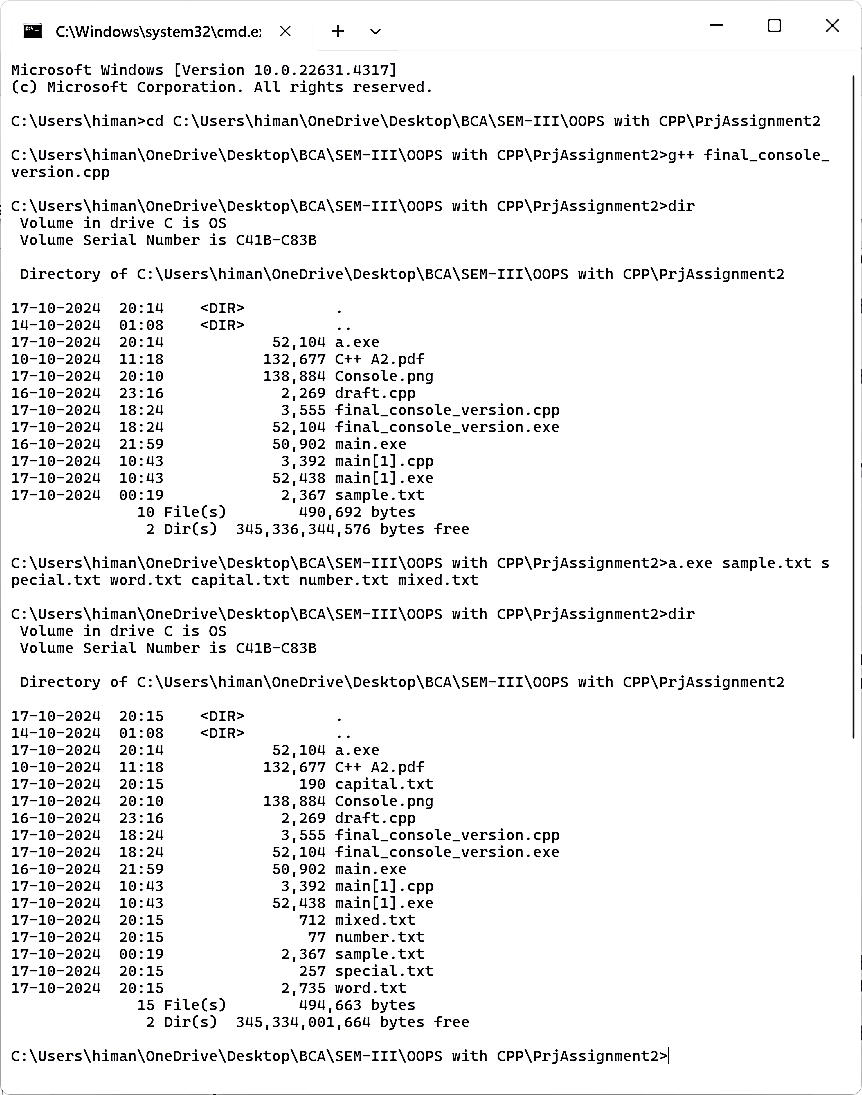
# Source Code

# Screenshots

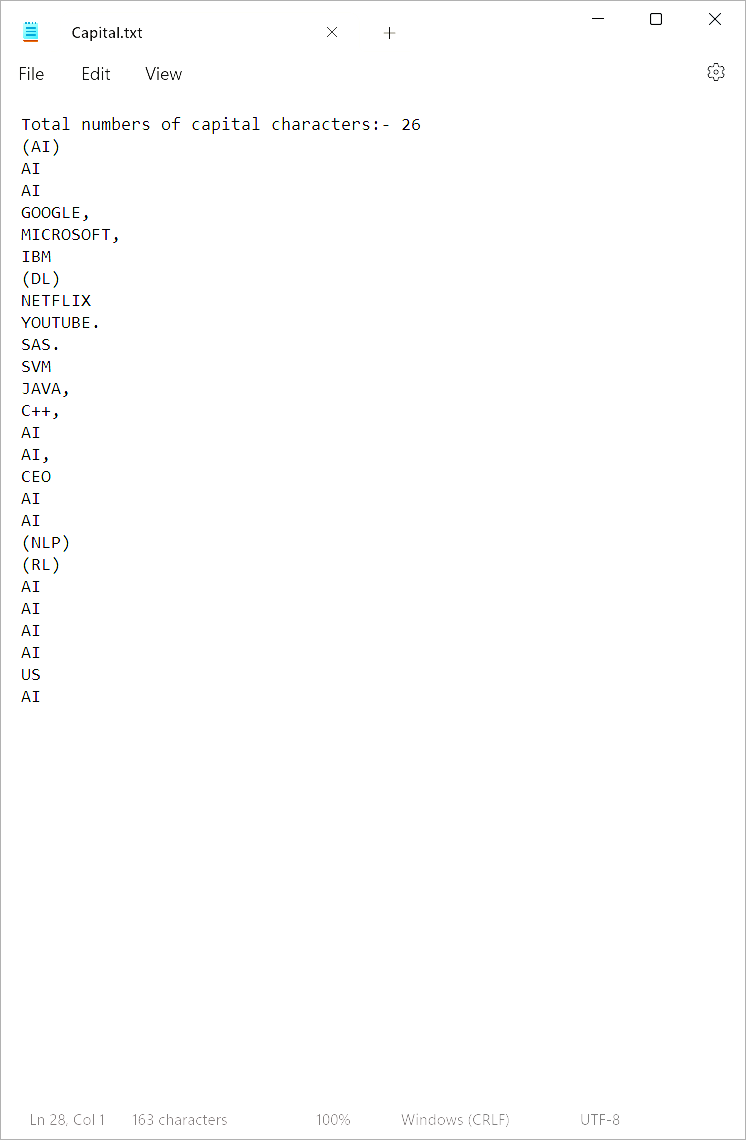
Data Set:

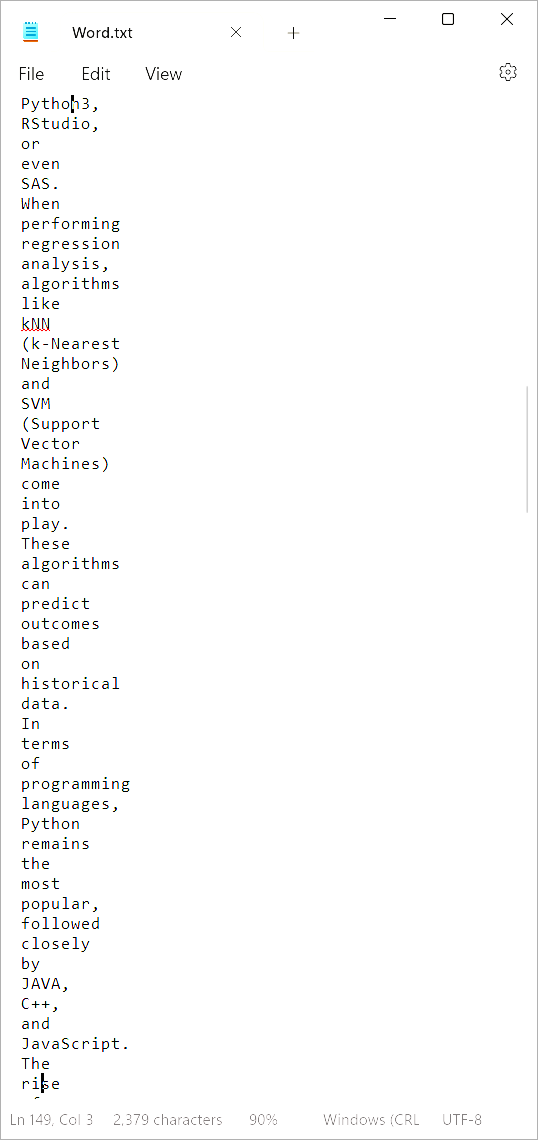
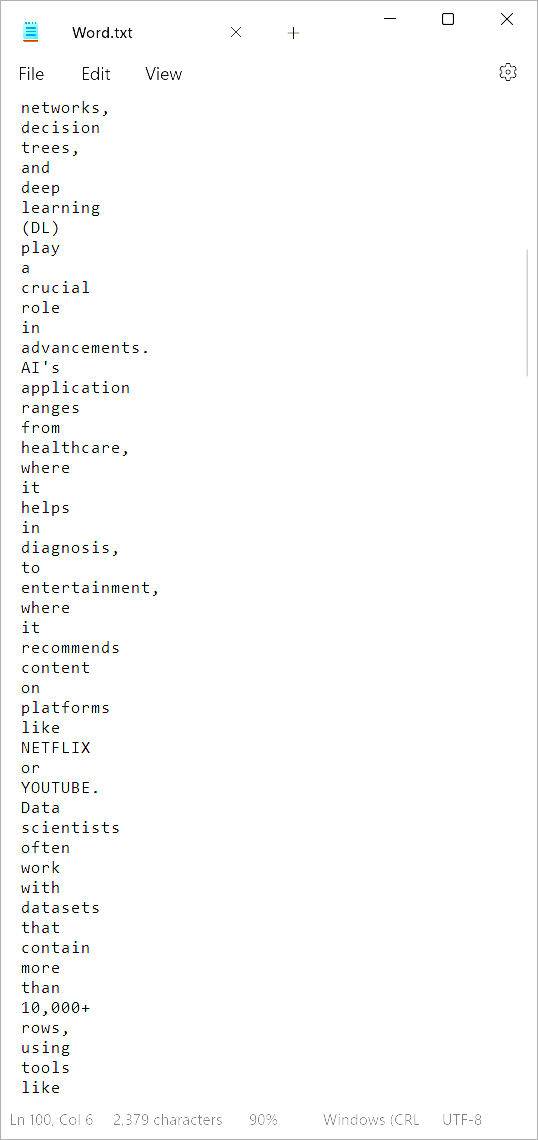
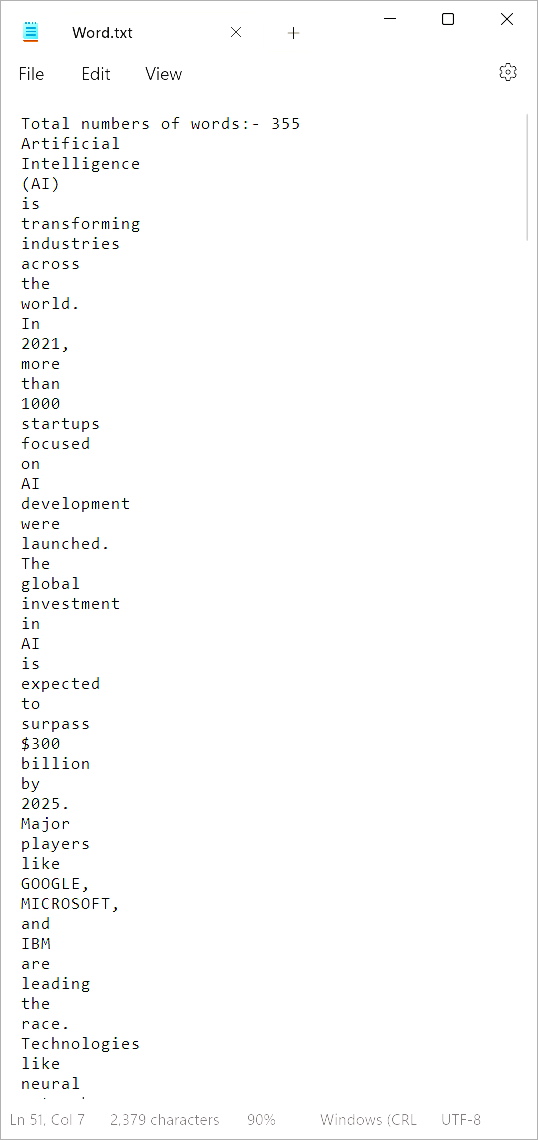
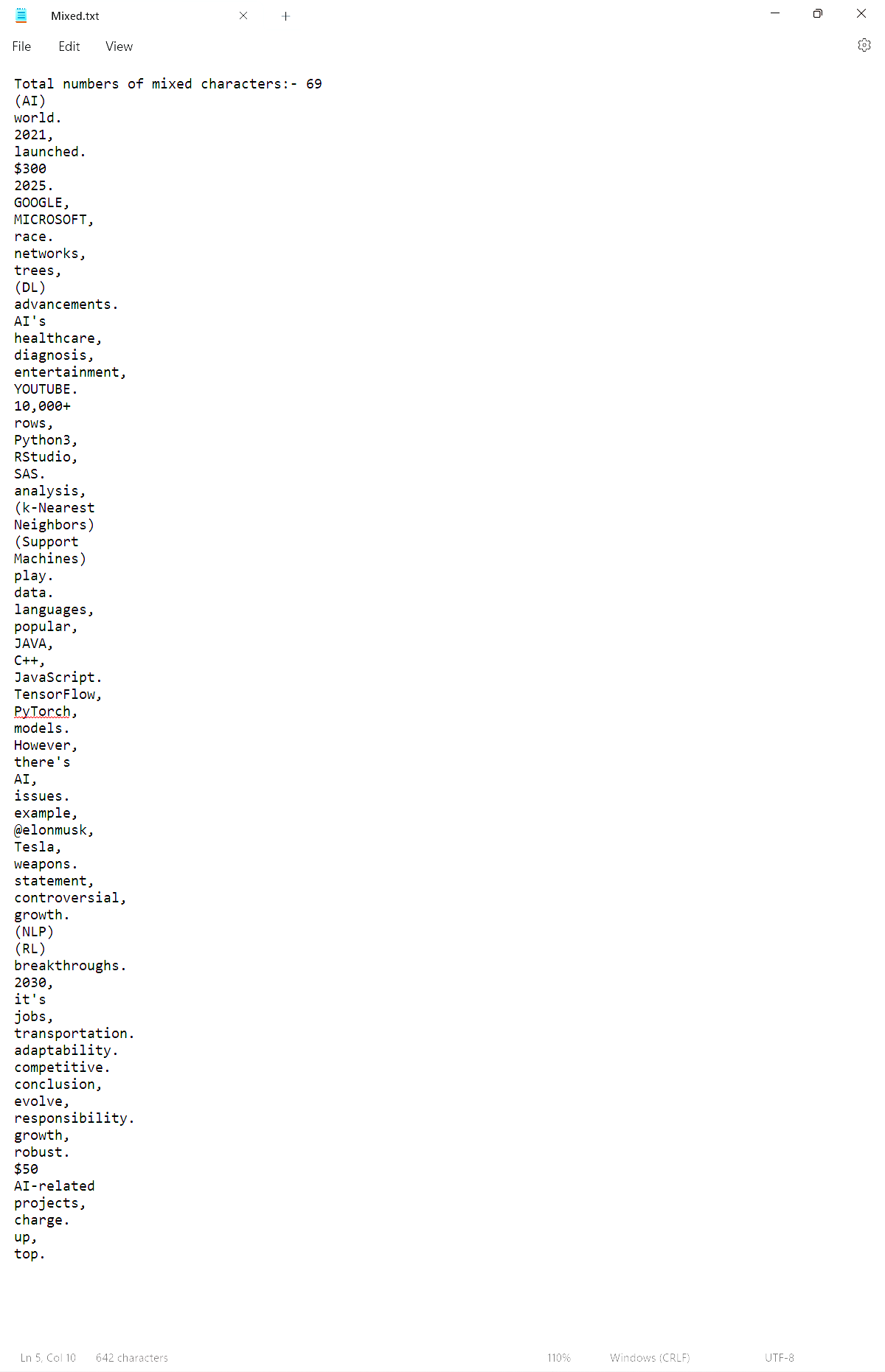


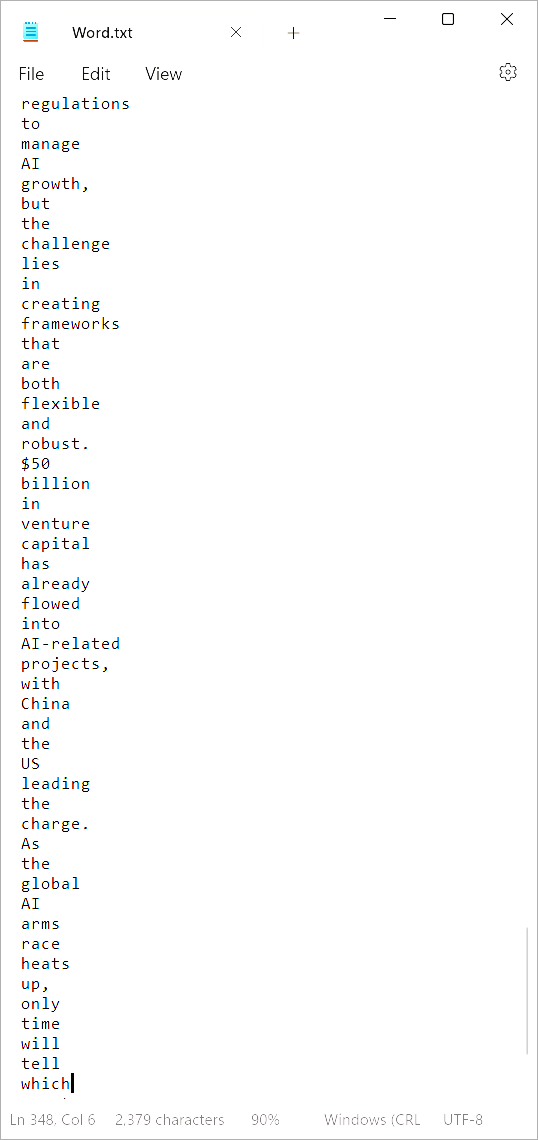
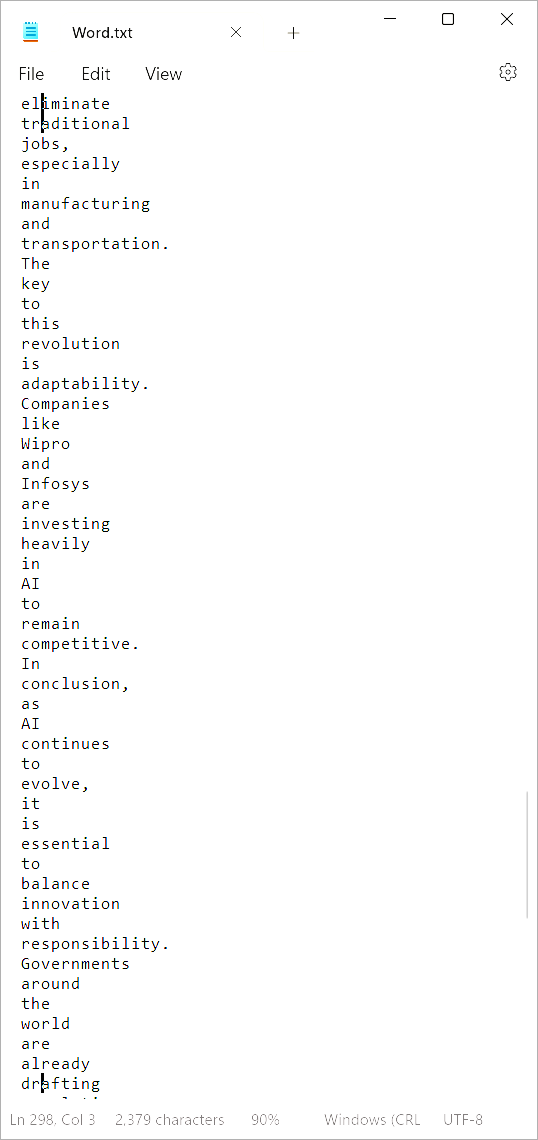
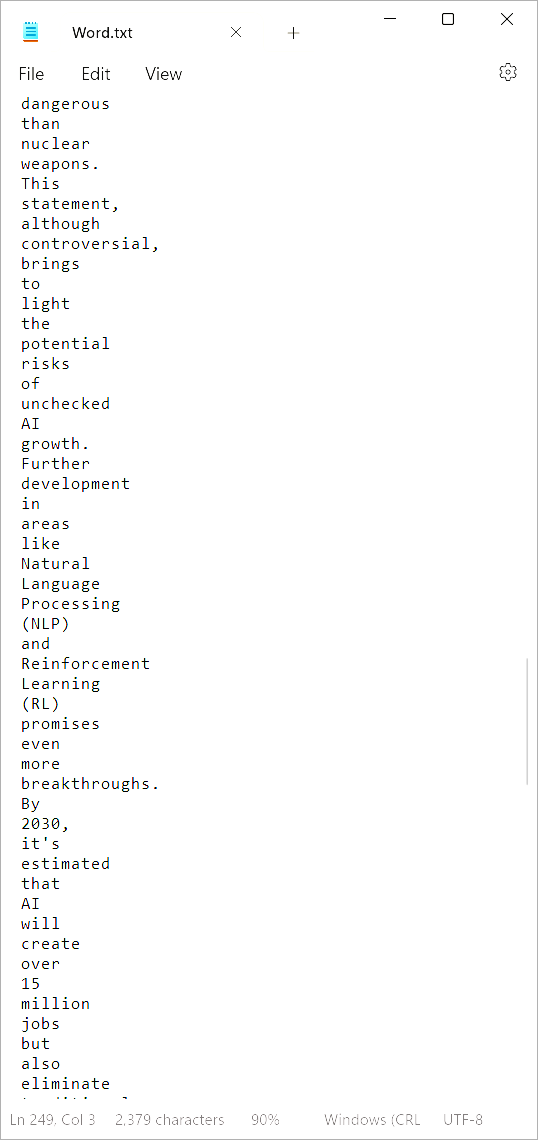
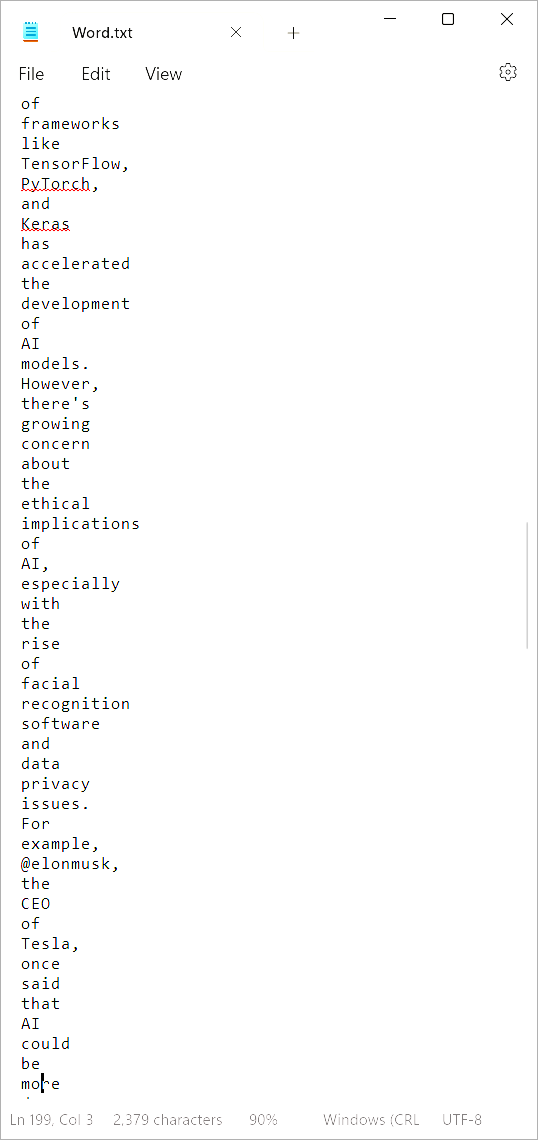
Console:

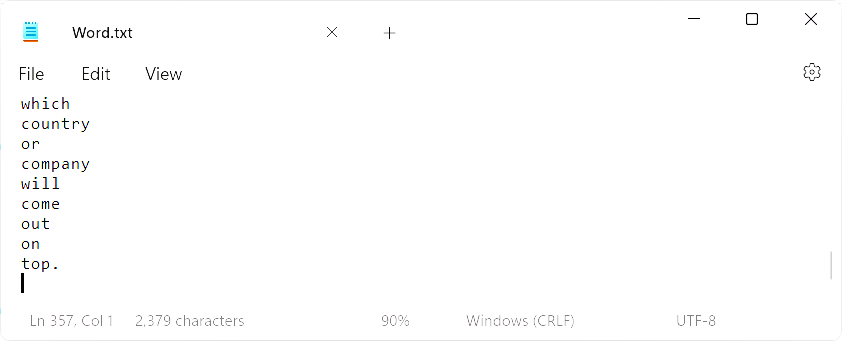


Output:









# Conclusion

The assignment allowed us to implement the very basic concepts of lexical analysis using C++. We implemented a program that reads a dataset, categorizes different types of data, and writes the results into separate output files to handle files and process data practically.

By leveraging command line arguments, we ensured that our program is versatile enough to seamlessly integrate with any dataset provided by the user, significantly enhancing its usability and accessibility.   
In organizing the results of our lexical analysis, we created structured output files, each displaying the total count of respective lexical units at the beginning, an essential feature for effective data interpretation and further analysis.

This project not only reinforced our technical skills in C++ but also emphasized the importance of proper data handling and processing.

This assignment provided valuable insights into lexical analysis and the powerful capabilities of C++ and it’s data handling capabilities.