Group Coursework (25%)

Cyberbullying / Toxic Text Analyzer in C

Background & Motivation

In today's digital society, communication increasingly takes place through online platforms such as social media, forums, and messaging applications. While these platforms enable connection and knowledge sharing, they have also become a breeding ground for harmful behaviors, including cyberbullying. Cyberbullying refers to the use of technology to harass, insult, or demean individuals, often through toxic or abusive language. Its impact can be severe, affecting mental health, social well-being, and even physical safety.

Detecting toxic or harmful language in text is an active research area within Natural Language Processing (NLP) and cybersecurity, with applications ranging from content moderation systems to protective tools for vulnerable users. However, most existing solutions rely on high-level languages and advanced machine learning libraries, which abstract away much of the fundamental programming work.

Project Overview

You are tasked with building a **Text Analyzer in C** that processes raw text datasets containing user-generated comments. Some comments may contain toxic or cyberbullying content (e.g., insults, threats, offensive language).

Your program must:

- Perform **basic (and advanced) text analysis** (counts, frequencies, sorting).
- Implement **linguistically inspired features** (stopword removal, sentiment approximations, n-gram detection).
- Provide a **"toxicity score"** per file based on suspicious word usage.

Objectives

At the end of this project, students are expected to demonstrate the ability to

- develop skills in string manipulation, arrays, structures and pointers in C
- perform file processing in C
- design and develop functions and modular program
- apply algorithmic thinking (frequency counts, sorting).

- apply computational techniques to a real-world cybersecurity + NLP context.
- design and develop a user-friendly user interface that facilitates easy navigation and interaction with the application, improving overall usability and accessibility for users.
- explain and present algorithmic thinking and computational techniques in documentations and comments, and in verbal presentation and demonstration.

Key Features

1. **Text Input and File Processing**
   - Load text input from one or more files (e.g., chat logs, comment files, or datasets provided). See suggested datasets section.
   - Support reading large text files line by line.
   - Handle multiple file types (e.g., .txt, or extracted .csv columns converted to .txt).

2. **Tokenization and Word Analysis**
   - Split sentences into words, normalizing them (case-insensitive, punctuation removal).
   - Count total words, unique words, and character statistics.
   - Identify and ignore common stopwords (e.g., “the”, “is”, “and”) using a provided stopword list.

3. **Toxic Word Detection**
   - Detect toxic words/phrases from a given lexicon (toxicwords.txt).
   - Count frequency of toxic terms and percentage of toxic words relative to the whole text.
   - Support expansion of lexicon by allowing students to update/extend toxicwords.txt.
   - Optionally categorize words by severity (e.g., mild, moderate, severe insults).

4. **Sorting and Reporting**
   - Sort words (alphabetically, by frequency, or by toxicity count) using at least one sorting algorithm (e.g., Bubble Sort, Quick Sort).
   - Display top N most frequent words and top N most frequent toxic words.
   - Provide summary statistics (e.g., number of sentences, average sentence length, lexical diversity index).

5. **Persistent Storage**
   - Save analysis results (word frequencies, toxic word counts, summary) into an output text file (e.g., analysis_report.txt).

- Allow re-loading of toxic/stopword dictionaries from file.
- Ensure reproducibility so the same text yields the same stored analysis.

6. User Interface

- Provide a simple menu-driven console interface, including options like:
    1. Load text file for analysis.
    2. Display general word statistics.
    3. Display toxic word analysis.
    4. Sort and display top N words.
    5. Save results to output file.
    6. Exit program.
- Include clear instructions and error messages for user guidance.

7. Error Handling & Robustness

- Gracefully handle invalid files or missing dictionary files.
- Manage empty or very large input files.
- Ensure robust handling of punctuation, numbers, and special characters.

Optional Bonus Extension Features:

- Crawl the web to collect raw toxic comments.
- Detect toxic bigrams/trigrams/phrases etc. (e.g., “shut up”, “go away”, “fxck off”, "go to hell", “worst human alive”).
- Implement hash-based word frequency tracking for efficiency.
- Compare two text files (e.g., benign vs toxic) and generate comparative reports.
- and the list goes on… (feel free to suggest)

Marking Scheme

	Criteria	**Core Features (Pass Level)**	**Advanced Features (Distinction Level)**	**Marks Allocation (/100)**
1	**Text Input & File Processing**	Load a single .txt file and process text line by line. Handle basic file errors (e.g., file not found).	Support multiple input files or larger dataset extracts. Handle edge cases (e.g., empty files, very large files).	10
		1 – 6 marks	7 – 10 marks	
2	**Tokenization & Word Analysis**	Split text into words, normalize (lowercase, strip punctuation), count total words and unique words.	Implement lexical diversity metrics (unique/total ratio), average sentence length, advanced statistics.	10
		1 – 10 marks	11 – 15 marks	
3	**Toxic Word Detection**	Detect toxic words using a provided dictionary (toxicwords.txt), count occurrences, report frequency.	Support multi-word toxic phrases (bigrams like “shut up”). Allow expansion of dictionary. Categorize words by severity (mild/moderate/severe).	20
		1 – 13 marks	14 – 20 marks	
4	**Sorting & Reporting**	Sort words alphabetically OR by frequency using a basic algorithm (e.g., Bubble Sort). Display top N frequent words.	Implement multiple sorting algorithms (Quick Sort, Merge Sort) and compare results. Display top toxic words separately.	10
		1 – 10 marks	11 – 15 marks	
5	**Persistent Storage**	Save analysis summary to an output file (analysis_report.txt). Reload stopword and toxic dictionaries from file.	Provide structured output (e.g., CSV or tabular format). Enable reproducibility by supporting multiple runs.	5
		1 – 6 marks	7 – 10 marks	
6	**User Interface**	Menu-driven text interface with options to load, analyze, display, save, and exit. Provide clear	Add enhancements such as ASCII bar chart for top words, comparative reports between files, or configurable menu	5

		user prompts.	options.	
		1 – 6 marks	7 – 10 marks	
7	**Error Handling & Robustness**	Handle invalid inputs gracefully (e.g., wrong filename). Prevent crashes on simple errors.	Anticipate more complex failures (e.g., corrupted files, unexpected encodings). Provide user-friendly error recovery.	5
8	**Code Quality & Modularity**	Use functions for modularity (e.g., loadFile(), countWords(), detectToxic()). Adequate commenting and readable style.	Advanced modularization (separating into multiple .c and .h files). Use of data structures (structs, arrays of structs, or hash tables) for clean design.	10
		1 – 6 marks	7 – 10 marks	
9	**Bonus Extension Features**	None	One or more features successfully done.	5
			5 marks	
10	**Presentation / Demo**	Video recording explaining the basic features and functions of the program. Video length is less than 5 min or longer than 10 min.	Video recording explaining clearly the features and functions of the program in relation to the code. Provided justification to the choice of techniques. Video length is within 5 – 10 min.	10
		1 – 6 marks	7 – 10 marks	
11	**Report / Documentation**	Students' documentation of document their approach, choices, and challenges, limitations and possible improvements are descriptive and lack of depth.	Students document their approach, choices, and challenges critically. Reflect on limitations and possible improvements critically.	10
		1 – 6 marks	7 – 10 marks	
	Total			**100**
	Module Assessment Allocation			**25%**

Suggested Datasets

1. **Dataset**: Jigsaw Toxic Comment Classification
   Source:
   - https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data
   - https://www.kaggle.com/competitions/jigsaw-multilingual-toxic-comment-classification/
   - https://huggingface.co/datasets/anitamaxvim/jigsaw-toxic-comments

2. **Dataset:** Toxigen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection.
   Source:
   - https://github.com/microsoft/ToxiGen
   - https://huggingface.co/datasets/toxigen/toxigen-data

3. **Dataset:** Cyberbullying Tweets Dataset
   Source: https://www.kaggle.com/datasets/andrewmvd/cyberbullying-classification

4. **Other datasets:**
   - https://pypi.org/project/toxic-comment-collection/
   - https://github.com/julian-risch/toxic-comment-collection

5. **Other relevant datasets** from other sources. You may also crawl social media platforms to collect your own raw data toxic comments.
   - Cite the source of the input toxic comment dataset in the report.
   - Please consult the module convenor if in doubt.

Deliverables

- **Source code** (.c and/or .h file(s)) with clear comments.
- **Input file**(s)
- **Output file**(s) including executable/binary of the program
- **Other** supporting files (if any)
- **Reflective report** (max 10 pages) covering:
  1. Cover page
  2. Key design choices (data structures, functions, sorting).
  3. Challenges faced (string handling, memory, efficiency).
  4. How you debugged and tested your program.
  5. Lessons learned about problem-solving in C.
  6. Readme instructions (how to set up and run the program)

7. Github link and the video recording link
8. Any other information

Instruction

1. This is group project of **3 or 4 members** per group. Only ONE member (normally the group leader) needs to perform the submission (can be helped by members).
2. Two modes of submission: **Moodle** and **Github**

   Mode of submission: **Moodle**

   - all relevant **source codes** in **.c, .h**
   - all **executable / binary** file in **.exe**
   - all other **supporting files**: object files (.o or .obj) (if any), static libraries (.a or .lib) (if any), dynamic libraries (.so or .dll) (if any) and any other files that are needed to support the running of your program.
   - a **reflective report** (**.docx AND .pdf**) with cover page, ReadMe instructions, GitHub link and the video recording link (follow the format of the sample template uploaded).

   Mode of submission: **Github**

   - Store a copy of all the files uploaded to Moodle on **Github** and provide the **Github link** in the submitted cover page.

Deadline: **5 December 2025 (Friday Week 14 before 17:00)**

Appendices (additional optional information for your reference)

Scaffolding / Staged Approach for Cyberbullying Text Analyzer in C

You are suggested to design and develop the solution to the coursework problem in stages:

Stage 1 – Foundation (Weeks 1–2)

Focus: Getting started with file handling and tokenization.

- Learn how to open, read, and process text files line by line.
- Implement basic tokenization: splitting words, removing punctuation, converting to lowercase.
- Count total words and unique words.
- Introduce stopword filtering using stopwords.txt.
- **Deliverable:** Small program that outputs word count, unique word count, and sample word frequency list from a given .txt file.

Learning outcomes: Basic I/O, string manipulation, arrays/structs.

Stage 2 – Core Analysis (Weeks 3–4)

Focus: Toxic word detection.

- Load a toxic word dictionary (toxicwords.txt).
- Scan processed tokens and detect toxic words.
- Count toxic words and calculate percentage of toxic content in the text.
- Update toxicwords.txt to extend the lexicon.
- **Deliverable:** Extended program that outputs toxic word statistics alongside general word stats.

Learning outcomes: Data representation, use of arrays/structs, modular programming with functions.

Stage 3 – Algorithmic Thinking (Weeks 5–6)

Focus: Sorting and reporting.

- Implement at least one sorting algorithm (e.g., Bubble Sort for beginners).
- Sort words by alphabetical order, frequency, or toxicity count.
- Display top N frequent words and top N toxic words.
- Introduce summary statistics: average sentence length, lexical diversity index.
- **Deliverable:** Program with interactive menu to choose analysis options.

Learning outcomes: Algorithm design, efficiency awareness, structured coding.

Stage 4 – Persistence & User Interaction (Weeks 7–8)

Focus: Storage and user interface.

- Add a **menu-driven console interface** with options (load, analyze, display stats, save, exit).
- Save analysis results into an **output report file (analysis_report.txt)**.

- Reload stopwords/toxic dictionaries dynamically.
- Ensure reproducibility (same input → same report).
- **Deliverable:** Fully interactive analyzer program with persistent storage.

Learning outcomes: Modular design, user experience in CLI programs, reproducibility.

Stage 5 – Robustness & Extensions (Weeks 9–10)

Focus: Error handling and advanced features.

- Gracefully handle missing files, empty files, and corrupted input.
- Add robust checks for unusual characters, numbers, encodings.
- Optional advanced features (for distinction-level students):
  - Toxic bigrams/trigrams (“shut up”, “go to hell”, “worst human alive”).
  - Efficiency: hash tables for word frequencies.
  - Comparative analysis: analyze two files (toxic vs benign) and compare results.
  - ASCII bar chart of top toxic words.
- **Deliverable:** Extended analyzer with robustness and at least one advanced feature.

Learning outcomes: Error resilience, design choices, optional research exposure to NLP/security themes.

Stage 6 – Reflection & Reporting (Week 11)

Focus: Research and educational reflection.

- Students document their approach, choices, and challenges (e.g., “I chose Bubble Sort because it’s simple, but realized it’s slow for large inputs”).
- Reflect on limitations and possible improvements.
- Insights on how the team reasons about algorithms, modularization, and robustness in C.
- **Deliverable:** Final report (2–3 pages) describing design, implementation, and learning reflection.

Learning outcomes*:* Metacognition, link coding work to efficiency/robustness/learning theory etc.

Student Progress Checklist

Instructions: Tick tasks as you complete them. Pass = Core, Merit = Enhanced, Distinction = Advanced.

Stage 1: Text Input & File Processing

- ☐ [Pass] Load and read a .txt file line by line
- ☐ [Pass] Count total words
- ☐ [Pass] Handle basic file errors
- ☐ [Merit] Support multiple input files
- ☐ [Merit] Handle large files efficiently
- ☐ [Distinction] Support CSV input (convert column to .txt)
- ☐ [Distinction] Optimize reading for very large datasets

Stage 2: Tokenization & Word Analysis

- ☐ [Pass] Tokenize words (normalize case, remove punctuation)
- ☐ [Pass] Count unique words
- ☐ [Pass] Ignore stopwords using stopwords.txt
- ☐ [Merit] Provide character-level stats (avg. word length)
- ☐ [Merit] Save filtered word list (excluding stopwords)
- ☐ [Distinction] Calculate lexical diversity index
- ☐ [Distinction] Add sentence-level stats (avg. sentence length)

Stage 3: Toxic Word Detection

- ☐ [Pass] Detect toxic words using toxicwords.txt
- ☐ [Pass] Count frequency and % toxic words
- ☐ [Merit] Allow dictionary updates (add new terms)
- ☐ [Merit] Print frequency of each toxic word
- ☐ [Distinction] Categorize toxic words by severity
- ☐ [Distinction] Detect multi-word toxic phrases (bigrams/trigrams)

Stage 4: Sorting & Reporting

- ☐ [Pass] Sort words alphabetically or by frequency (Bubble Sort)
- ☐ [Pass] Display top N frequent words
- ☐ [Merit] Implement another sorting method (Quick/Merge)
- ☐ [Merit] Display top N toxic words separately
- ☐ [Distinction] Compare multiple sorting algorithm outputs
- ☐ [Distinction] Generate extra summary stats (toxic ratio vs non-toxic)

Stage 5: Persistent Storage & User Interface

- ☐ [Pass] Save results to analysis_report.txt
- ☐ [Pass] Provide a menu-driven interface
- ☐ [Merit] Reload stopword/toxic dictionaries dynamically
- ☐ [Merit] Save results in structured tabular format (CSV)
- ☐ [Distinction] Add ASCII bar chart for top toxic words
- ☐ [Distinction] Compare two files (benign vs toxic analysis)

Stage 6: Error Handling & Robustness

- ☐ [Pass] Handle invalid input filenames
- ☐ [Pass] Prevent crashes for empty files
- ☐ [Merit] Handle unusual characters and numbers robustly
- ☐ [Merit] Provide clear error messages
- ☐ [Distinction] Handle corrupted/large files gracefully
- ☐ [Distinction] Add user-friendly recovery instructions

Stage 7: Reflection & Reporting

- ☐ [Pass] Write short implementation report
- ☐ [Merit] Include design choices (e.g., sorting rationale)
- ☐ [Distinction] Link reflection to efficiency/robustness/learning theory

End of Question – All the best!