1.0 Introduction

This project is designed to generate unique three-letter abbreviations for a list of names, adhering to some specific rules as outlined by the course instructor. The primary challenges included to ensure abbreviations are:

1. Formed from the first letter of the name followed by two additional letters in order.
2. Unique across all names in the list.
3. Scored and selected based on predefined positional and frequency rules.

This report outlines the program's structure, the problem-solving approach, and the validation process used to ensure its accuracy.

2.0 Program Design

Structure

The program is implemented in Python, following a modular and logical structure to ensure clarity and maintainability. Key functions include:

1. **load_names(filename):** Reads names from an input file, ensuring the list is cleaned of empty or invalid lines.
2. **load_letter_values(filename):** Loads scoring values from a predefined file (values.txt) for computing abbreviation scores.
3. **generate_abbreviations(name):** Generates all possible three-letter abbreviations from a name by combining the first letter with two subsequent letters in order.
4. **calculate_score(abbreviation, words, values):** Computes the score for each abbreviation based on:
   - Position within the name (start, middle, or end).
   - Frequency-based scoring for each letter.
5. **find_best_abbreviation(names, values):** Identifies the best abbreviation(s) for each name, ensuring duplicates across names are excluded and only the lowest score(s) are selected.
6. **main():** Integrates all functions to load input files, process names, and generate the final output file.

Key Algorithms

1. **Abbreviation Generation**: Each name is split into valid words. The first letter of the name is combined with any two subsequent letters to form potential abbreviations.
2. **Scoring**: Abbreviation scores are calculated as follows:
   - The first letter always has a score of 0.
   - Last letters of words score 5 (or 20 if the letter is E).
   - Middle letters score a combination of their position and frequency-based values.
3. **Selection**: Abbreviations that appear in multiple names are excluded. For each name, the abbreviation(s) with the lowest score is selected.

3.0 Testing and Validation

Test Cases

1. **Basic Input**:
   - Input: Cold, Cool, C++ Code.
   - Expected Output: Cold: CLD, Cool: COO, C++ Code: CCD
2. **Edge Cases**:
   - Single-letter names (e.g., A): No valid abbreviation.
   - Names with repeated letters (e.g., C++ Code): Correct handling of repeated letters based on position.

3. **Complex Input**:
    - Tested using trees.txt, a list of British native trees, which produced accurate and detailed results, including tied abbreviations where applicable.

Validation Methods

1. **Debug Logging**: Detailed debug logs were enabled during testing to track:
    - Abbreviations generated for each name.
    - Scores calculated for each abbreviation.
    - Final abbreviations selected.
2. **Manual Cross-Checking**: Specific examples were manually checked, such as:
    - Cold generated CLD with a score of 22.
    - C++ Code generated CCD with the lowest score of 11.
3. **Comparison with Expected Outputs**: Outputs were compared with expected results from the assignment brief to ensure correctness.

	Input	**Expected output**	**Actual output**
1	Tableau	Tableau TBU	Tableau TBU
2	Amazon	Amazon AAZ	Amazon AAZ
3	Azure	Azure AZR	Azure AZR
4	Machine	Machine MCH	Machine MCH
5	Microsoft	Microsoft MFT	Microsoft MFT
6	Note Pad	Note Pad NPD	Note Pad NPD
7	Tool Kit	Tool Kit TKT	Tool Kit TKT
8	Python Window	Python Window PWW	Python Window PWW
9	Product Sub-Category	Product Sub-Category PCC PCS PSC	Product Sub-Category PCC PCS PSC
10	Equiconvex	Equiconvex EQE	Equiconvex EQE
11	OBJECT ORIENTED PROGRAMMING	OBJECT ORIENTED PROGRAMMING OOO OOP OPO	OBJECT ORIENTED PROGRAMMING OOO OOP OPO
12	DATA ENGINEERING	DATA ENGINEERING DEE	DATA ENGINEERING DEE
13	Cold	Cold CLD	Cold CLD
14	Cool	Cool COO	Cool COO
15	C++ Code	C++ Code CCD	C++ Code CCD

4.0 Conclusion

The Program

- Generates abbreviations that are unique, scored, and selected based on the lowest score.
- Handles edge cases and large inputs efficiently.
- Validates through debug logs and manual cross-checking ensured accuracy.