User Guide: Zip Code Data Processing System

AUTHORED BY:

Adams Aidan
Chermack Brendan
Eze Daniel
Hoerger Thomas
Mohammed Mujahid
Verweg Matthew

Table of Contents

1. Introduction

The **Zip Code Data Processing System**, an efficient tool for reading, sorting, converting, and indexing large datasets of Zip Code records. This system processes **CSV files** containing location data, converts them into **length-indicated formats**, and generates **index files** for fast lookups.

This guide will walk you through the components, setup, usage, and troubleshooting of the project.

2. System Overview

This system reads **Zip Code data** from CSV files, organizes it by state, identifies **extreme points** (northernmost, southernmost, etc.), and outputs the data into structured CSV files. It also supports **length-indicated formats** for more efficient parsing of variable-length records and **index files** for quick access.

3. Installation and Setup

1. **Pre-requisites**:
    a. C++ compiler (e.g., g++)
    b. Make sure your development environment supports **C++17** or later (VS Code is highly recommended).
    c. **CSV input files** (e.g., us_postal_codes.csv).

2. **Cloning the Repository (optional, this step is required only if the code isn't present on your host machine)**:

    If the code is hosted in a repository, clone it using the following else skip to step 3:

    git clone <repository-url>
    cd <project-directory>

3. **Compiling the Code**:

    Use the following command to compile all source files:

    g++ -o main main.cpp Buffer.cpp CSVProcessing.cpp CSVLengthIndicated.cpp IndexFile.cpp

4. **Running the Program**:

    Execute the program:

    ./main

4. How to Use the System

4.1 Running the Main Application

The **main application** is controlled through main.cpp, which coordinates the **processing, conversion, and indexing** of Zip Code data. Once compiled, run the program from your terminal:

./main

This will generate:

- **Sorted CSV files** with headers.
- **Length-indicated files** (if enabled).
- **Index files** for fast lookups.
- **Search** length indicated list with fast lookups.

4.2 CSV File Processing

1. Place your **CSV files** (e.g., us_postal_codes.csv) in the project directory.
2. The system processes these files and **groups Zip Codes by state**.
3. Each state's **northernmost, southernmost, easternmost, and westernmost Zip Codes** are identified.

Output CSVs:

- **output1.csv** and **output2.csv** will be generated, each with headers and sorted data.

4.3 Length-Indicated File Conversion

The system supports **converting CSVs into length-indicated ASCII files**. This conversion adds a **two-digit length prefix** to each field for efficient reading.

Steps:

1. Modify main.cpp to **enable length-indicated conversion** (uncomment the relevant lines).
2. Run the program to generate a **length-indicated version**:
3. us_postal_codes_length_indicated.csv

4.4 Index File Generation

The **IndexFile class** generates index files to map **Zip Codes to offsets** in the length-indicated file.

Steps:

1. Ensure you have the length-indicated file (e.g., us_postal_codes_length_indicated.csv).
2. Run the program to generate the index file: index.txt
   This file contains: <zip_code> <offset>

4.5 Indexed File search

Index File search is done in main. This uses an indexed file and inbuilt functions. The functions parse the index file for the wanted zip codes. Which are entered in in this format –zZipcode1-zZipcode2 … etc. Once the zip codes are entered in the offset is grabbed from the index. Then the length indicated list is searched with the offset for the wanted data.

5. Input and Output Files

- **Input Files**:
  - us_postal_codes.csv: A CSV containing Zip Code, state, and location details.
- **Output Files**:
  - output1.csv and output2.csv: CSVs with headers and sorted data.
  - us_postal_codes_length_indicated.csv: Length-indicated version of the original CSV.
  - index.txt: Index file mapping Zip Codes to offsets.

6. Class Components Explained

6.1 Buffer Class

- **Responsibilities**:

Reads **CSV files** and **length-indicated binary files**.
Groups records by **state** for easy access.

- **Usage**:
  Call read_csv() to load records into memory.
  Use get_state_zip_codes() to access data grouped by state.

6.2 CSVProcessing Class

- **Responsibilities**:
  Sorts Zip Codes by state.
  Identifies extreme points (northernmost, southernmost, etc.).
  Generates CSV output with headers.
- **Usage**:
  Call sortBuffer() to sort data and identify extremes.
  Use csvOutput() to generate the final CSV.

6.3 CSVLengthIndicated Class

- **Responsibilities**:
  Converts CSV data into **length-indicated format**.
  Reads length-indicated files and extracts data.
- **Usage**:
  Use convertCSVToLengthIndicated() to generate the length-indicated version.

6.4 IndexFile Class

- **Responsibilities**:
  Generates **index files** mapping Zip Codes to offsets.
- **Usage**:
  Call createIndexFile() to produce the index file.

7. Error Handling

The system includes **basic error handling** for:

- **File I/O errors**:

If a file fails to open, the program will print an error message.

- **Invalid data formats**:

The program will notify if **latitude/longitude values** are invalid.

8. Troubleshooting

1. **CSV File Not Found Error**:
    a. Ensure your **CSV files** are in the same directory as the program.
    b. Verify the **file name and extension** (e.g., us_postal_codes.csv).
2. **Index File Generation Fails**:
    a. Ensure the **length-indicated file** exists before generating the index.
    b. Check that the **length-indicated file** is formatted correctly.
3. **Program Crashes or Freezes**:
    a. Check for **missing or corrupted input files**.
    b. Verify that the **CSV files contain valid data** (e.g., numeric latitude/longitude).

9. Conclusion

This **Zip Code Data Processing System** provides a flexible, modular way to process large datasets of Zip Code information. By **sorting data, converting formats, and generating index files**, the system ensures efficient access and easy management of location-based data.

This guide covers:

- **How to run the program**
- **How to process and convert files**
- **How to generate index files**