**Project 2: Zip Code — Sequential Files, Header Records, and Indexing**

**Team 4:** Minh Quan Tran, Mitchell Rogers, Carson Kariniemi, Abel Asfaw, Mahad Farah  
**Instructor:** Andrew Anda  
**Due Date:** 10/22/2025

**Design Document**

**1. Introduction**

This project extends Project 1 by adding (1) support for **sequential processing** of CSV data, (2) a **length-indicated header-record** variant, and (3) a **primary-key index** with command-line ZIP lookups.

The codebase provides:

* Baseline CSV ingestion and display (sorted by ZIP or by State).
* A header-record format where each logical row includes a leading record length for demonstration.
* A primary key indexer/lookup program that builds or loads a binary index mapping ZIP → file offset and supports CLI searches via flags like -Z56301.

**2. Data Structures**

**2.1 PostalCodeItem**

**Files:** PostalCodeItem.h, PostalCodeItem.cpp  
Represents a single CSV postal entry.

**Fields**

* int zip
* std::string place
* std::string state
* std::string county
* double latitude
* double longitude

**Responsibilities**

* Encapsulate one postal record.
* Provide getters/setters and formatted output via printInfo().

**2.2 PostalList**

**Files:** PostalList.h, PostalList.cpp  
Manages a collection of PostalCodeItem.

**Fields**

* std::vector<PostalCodeItem> items

**Responsibilities**

* Add items (addItem), access items (getItem, findByZip), report size.
* Print all entries or sorted views:
  + printSortedByZip()
  + printSortedByState() (state, then ZIP)

**2.3 HeaderRecordPostalCodeItem**

**Files:** HeaderRecordPostalCodeItem.h, HeaderRecordPostalCodeItem.cpp  
Variant of a postal record that includes a **record length** field to model a length-indicated format.

**Fields**

* int recordLength
* int zip
* std::string place
* std::string state
* std::string county
* double latitude
* double longitude

**Responsibilities**

* Store a record length along with standard postal fields.
* Provide getters/setters + printInfo() with the length printed as the first column.

**Note:** In the current parser, recordLength is read from the first **two** characters of each line (see §4.2), not from a binary length field.

**2.4 HeaderRecordPostalList**

**Files:** HeaderRecordPostalList.h, HeaderRecordPostalList.cpp  
List container for HeaderRecordPostalCodeItem, mirroring PostalList with the same sorted print utilities.

**3. Input Modules**

**3.1 CSV Reader (plain)**

**File:** readCSV.cpp  
**Function:** void inputCSVtoList(PostalList& inputList, std::string fileName)

**Behavior**

* Opens a plain CSV (zip,place,state,county,latitude,longitude).
* Skips the first header line.
* Parses each subsequent line by naive comma splitting (find(',') + substr).
* Constructs a PostalCodeItem and appends it to PostalList.

**Assumptions**

* No quoted fields, no commas inside fields, and all lines are well-formed.

**3.2 Header-Record Reader (length-indicated text)**

**File:** readHeaderRecordPostalCodeBuffer.cpp  
**Function:** void inputCSVtoList(HeaderRecordPostalList& inputList, std::string fileName)

**Behavior**

* Opens a **length-indicated textual** file where each data line begins with a 2-digit length, then a comma, then the CSV payload.
* Skips the first header line.
* For each data line:
  + recordLength = stoi(line.substr(0, 2))
  + line = line.substr(3, ...) (skips the 2 digits and the following comma)
  + Parses remaining fields by commas into HeaderRecordPostalCodeItem, then adds to the list.

**Assumptions**

* The first two characters always encode a **two-digit** length (00–99) followed by a comma.
* No embedded commas/quotes within fields.

This is a **demonstration** of a length-indicated record. It is text-based and fixed to two digits; future versions can adopt binary 4-byte lengths and remove the 99-byte cap.

**4. Programs**

**4.1 main1.cpp — Sorted by ZIP (plain CSV)**

* Input: us\_postal\_codes.csv
* Uses PostalList + readCSV.cpp
* Prints a labeled table and calls printSortedByZip().

**4.2 main2.cpp — Sorted by ZIP (row-randomized CSV)**

* Input: us\_postal\_codes\_ROWS\_RANDOMIZED.csv
* Same pipeline as main1.cpp.
* Confirms that row order in the CSV does not affect sorted output.

**4.3 main3.cpp — Sorted by ZIP (length-indicated header records)**

* Input: us\_postal\_codes\_length\_indicated\_header\_record.txt
* Uses HeaderRecordPostalList + readHeaderRecordPostalCodeBuffer.cpp
* Prints the first column as the header “HR” (record length) and sorts by ZIP.

**4.4 main4.cpp — Sorted by ZIP (row-randomized, length-indicated)**

* Input: us\_postal\_codes\_ROWS\_RANDOMIZED\_length\_indicated\_header\_record.txt
* Same pipeline as main3.cpp to confirm order independence.

**4.5 indexer\_lookup.cpp — Primary Key Index + CLI Lookup**

**Files:** single translation unit (as pasted)

**Behavior**

* Constants:
  + DATA\_FILE = "us\_postal\_codes.csv"
  + INDEX\_FILE = "indexfile.bin"
* If indexfile.bin exists:
  + Loads a binary index into std::unordered\_map<std::string, std::streampos>.
  + Binary format per entry:  
    uint8\_t zipLength, <zipLength bytes of zip>, uint64\_t offset
* Else:
  + Builds the index by scanning us\_postal\_codes.csv (skipping header):
    - For each line, stores offset = tellg() and zip = substring before first comma.
  + Writes indexfile.bin in the format above.
* Command line:
  + Accepts flags -Z<zip> (e.g., -Z56301 -Z99999).
  + For each requested ZIP:
    - If not in index → prints “<zip> not found.”
    - If present → seeks to offset, reads line, prints fields with labels.

**Purpose**

* Demonstrates an **external primary key index** and in-place random access to a single target line, satisfying the constraint that the full data set is not loaded into RAM.

**5. Core Algorithms**

**5.1 Plain CSV Load**

1. Open CSV, getline the header (discard).
2. For each line:
   * Repeatedly find , and substr field values.
   * Convert numeric fields via stoi/stod.
   * Construct PostalCodeItem, push to PostalList.

**5.2 Length-Indicated Header Record Load (Textual)**

1. Open .txt, getline the header (discard).
2. For each line:
   * Parse recordLength = stoi(line.substr(0, 2)).
   * Drop the length and following comma: line = line.substr(3, ...).
   * Split on commas as in plain CSV.
   * Construct HeaderRecordPostalCodeItem (including recordLength), push to HeaderRecordPostalList.

**5.3 Index Build / Load / Lookup**

* **Build** (if no index):
  + Iterate through us\_postal\_codes.csv, skip header.
  + Before reading each line, capture pos = tellg().
  + Extract ZIP as substring before first comma.
  + Store map[zip] = pos.
  + Write out indexfile.bin as (zipLen, zipBytes, offset).
* **Load** (if index exists):
  + Read entries until EOF; populate unordered\_map.
* **Lookup**:
  + For each -Zxxxx, if map contains xxxx:
    - seekg(offset), getline() one record, print labeled fields.
  + Else print “not found”.

**6. Error Handling & Edge Cases**

* File open failures are reported to std::cerr.
* CSV readers assume well-formed lines (no quotes, no embedded commas); malformed lines may throw on stoi/stod or produce incorrect splits.
* The length-indicated text reader assumes a **two-digit** length and a comma delimiter after the length; longer payloads will truncate the declared length or be inconsistent if the prefix is incorrect.
* Index reader assumes the index file matches the current CSV; if the CSV changes, the index may be stale (not currently detected).

**7. Assumptions & Constraints**

* Input CSVs have header lines and the canonical 6 data fields in order.
* No quoting or escaped commas inside fields.
* The “length-indicated” files are a textual teaching format: LL,<csv...> where LL is two digits. (Not a binary length prefix.)
* For the index demo, only a **single record** is read into RAM at lookup time; the full file is not loaded (satisfies the “single-record in RAM” constraint).
* Sorting is performed on in-memory containers (vector), which is acceptable for the display use-case in this assignment.

**8. Build & Run**

**Build (example):**

g++ -std=c++17 -O2 -o main1 main1.cpp PostalList.cpp PostalCodeItem.cpp readCSV.cpp

g++ -std=c++17 -O2 -o main2 main2.cpp PostalList.cpp PostalCodeItem.cpp readCSV.cpp

g++ -std=c++17 -O2 -o main3 main3.cpp HeaderRecordPostalList.cpp HeaderRecordPostalCodeItem.cpp readHeaderRecordPostalCodeBuffer.cpp

g++ -std=c++17 -O2 -o main4 main4.cpp HeaderRecordPostalList.cpp HeaderRecordPostalCodeItem.cpp readHeaderRecordPostalCodeBuffer.cpp

g++ -std=c++17 -O2 -o zipindex indexer\_lookup.cpp

Run examples:

./main1

./main2

./main3

./main4

# build/load index and search

./zipindex -Z56301 -Z99999

**9. Design Decisions & Rationale**

* **Separation of concerns:** Item vs. List vs. Reader modules keep responsibilities focused and testable.
* **Two representations:** PostalCodeItem/PostalList for plain CSV; HeaderRecordPostalCodeItem/HeaderRecordPostalList for length-indicated records.
* **Indexing format:** Simple binary file (zip length + zip bytes + 64-bit offset) balances simplicity and speed.
* **Memory model:** For lookups, only the header (skipped), index map, and one line buffer are resident.

**10. Known Limitations & Future Work**

* **CSV robustness:** Current parsing assumes no quotes or embedded commas. Future: adopt an RFC-4180 compliant splitter.
* **Length-indicated format:** Presently a **textual 2-digit prefix**. Future: upgrade to a **binary 4-byte** little-endian length field with a proper header that records length format (ASCII/Binary), length size, and whether the length includes itself.
* **Header schema:** Consider adding a real **header record** that lists field names/order and tracks which field is the primary key (to survive column re-ordering).
* **Stale index detection:** Add a header checksum/timestamp and store it in the index to detect mismatch.
* **Build hygiene:** Replace #include "\*.cpp" in headers with normal compilation units and proper include guards; this will reduce ODR/link issues.

**11. Doxygen & Documentation**

* Each class/file includes Doxygen comments (@file, @brief, authors, dates).
* Suggested structure for Doxyfile:
  + RECURSIVE = YES
  + EXTRACT\_PRIVATE = NO
  + GENERATE\_LATEX = YES
  + Build PDF via generated latex/ → make.
* Consider adding two Doxygen pages:
  + \page user\_guide User Guide — build/run instructions and CLI examples.
  + \page design\_overview Design Overview — summarize modules, file formats, and future work.

**12. Conclusion**

This codebase delivers the core outcomes of Project 2:

* Sequential CSV processing with sorted displays (ZIP/State).
* A teaching example of **length-indicated** records (textual prefix).
* A working **primary-key index** with command-line queries that reads only a **single record** into RAM at lookup time.

The design remains modular and forward-compatible with more robust file formats (true length-prefixed binary), schema headers, and richer indexing—paving the way for a complete indexed-sequential file system in subsequent assignments.